API Guide

VolServ Version 5.0
September 2001
601355 Rev A

Trademark Notice

AMASS, DataMgr, EMASS, FileServ, and VolServ are either trademarks or registered trademarks of ADIC, Advanced Digital Information Corporation. DAS is a trademark of Grau, an ADIC subsidiary. All other product names and identifications are trademarks or registered trademarks of their respective manufacturers.

Copyright Notice

ADIC
10949 East Peakview Ave.
Englewood, CO 80111 USA
Phone: 303-792-9700
FAX: 303-792-2465

U.S. Government Rights Restricted

Use, duplication, or disclosure of either the software or documentation is subject to restrictions set forth by the U.S. Government in FAR 52.227-19(c)(2) and subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and/or in similar or following clauses in the FAR, DoD, or NASA FAR Supplement.

Technical Assistance

ADIC Technical Assistance Center:

- In the USA and Canada, call 1-800-827-3822.
- Outside the USA and Canada, call 303-874-0188 or toll-free 00800-9999-3822.
- Send e-mail to: support@adic.com.

Documentation

Although the material contained herein has been carefully reviewed, ADIC does not warrant it to be free of errors or omissions. We reserve the right to make corrections, updates, revisions, or changes to the information contained herein.

- Send e-mail to: techdocs@adic.com

READER COMMENT FORM

ADIC includes this Form in an effort to provide the best possible documentation to our customers. Please take a few moments to mail or FAX your response to:

ADIC
Software Documentation
10949 East Peakview Ave.
Englewood, CO 80111
FAX: 303-792-2465
E-mail: techdocs@adic.com

Question	Circle One	
Information was complete.	Agree	Disagree
Information was easy to find.	Agree	Disagree
Information was easy to follow.	Agree	Disagree

Is there anything you especially *like* or *dislike* about the organization, presentation, or writing in this manual?_____

_____

_____

_____

_____

Book Title	Document Number
Customer Name	Telephone
E-mail Address	
Company Name	
Address	
City, State, Zip	

NOTES

Contents

Preface

Getting Started 1

API Functions 2

Preface

NOTES

Purpose of This Book

This book describes the VolServ application programming interface (API). The API consists of functions, iterators, symbolic names, type definitions, and data structures.

Using the API provides the programmer with the ability to directly manipulate VolServ file system metadata and media.

Who Should Read This Book

This book is written for programmers who are creating or modifying an application that requires tight control over data in the file system managed by VolServ

How This Book is Organized

This book contains the following chapters:

Chapter 1: Getting Started — Describes API types and naming conventions, dispatch routines and global parameters,.and error handling.

Chapter 2: Functions — Alphabetical list of API function calls.

Appendix A: Valid Status Fields — A matrix shows which status fields are valid for each command.

Appendix B: Error Codes.

Appendix C: Mount Example.

Conventions

The conventions used throughout the VolServ technical books are listed below:

Convention	Example

Screen text, file names, program names, and commands are in `Courier` font.	`Request to add a new volume:`
	`Volume group will be “20”`
	`Volume position will be “A123”.`

The root prompt is shown as a number symbol.	`#` **`su root`**
What you should type in is shown in `Courier` **bold** font.	**`vsarchiveqry`**
Site-specific variables are in a *Times italics* font.	**tar -xvf** ***tapedevicename***
A backward slash (\ ) denotes the input is continued onto the next line; the printed page is just not wide enough to accommodate the line.	# **remsh** ***nodename*** **-n dd if=/dev ** **/*tapedevicename*/bs=20b | tar xvfb ** **- 20** (You should type the entire command without the backward slash.)
Pressing <Return> after each command is assumed.	
A menu name with an arrow refers to a sequence of menus.	Config-->MediaType-->Redefine

Books

The books described below are part of the technical documentation set, and are shipped on CD along with the VolServ software:

Overview
Provides an overview of VolServ. Contains a glossary.

Installing VolServ
Describes server requirements, installation instructions, troubleshooting procedures, and configuration parameters.

Using the VolServ GUI
Describes how to perform system administrative tasks using the graphical user interface.

API Guide
Provides a list of API functions.

Administrative Tasks
Describes how to perform system administrative tasks using VolServ commands.

Command Reference
Contains a list of VolServ commands

Error Messages
Provides corrective action for system log errors.

Quick Reference Card
Summarizes commands.

Online Books

The documentation CD contains VolServ book files and Adobe® Acrobat® Reader. The Reader allows you to view and navigate the online documentation files yet preserves the page design and graphics from the printed books.

Related Publications

The publications described in the table below are created and distributed on an as-needed basis.

Related Publications	Description
"Release Notes"	For each version of VolServ, the "Release Notes" contain: • Summary of enhancements. • Describes: - Fixed problems. - Known problems. - Installation and configuration issues. • Lists: - Operating system patches. - System requirements.
"Product Alerts"	Informs customers of technical problems and solutions.
"Product Bulletins"	Conveys technical information—not problems—to customers.

Contact Publications Department

To make corrections or to comment on VolServ publications, please contact Software Technical Publications at our e-mail address: techdocs@adic.com.

Secured Web Site

To receive access to the secured site on our home page containing technical product information (Release Notes, Product Alerts, Product Bulletins, FAQs), visit http://partners.adic.com/ and follow the password request procedure. In return, ADIC will send you instructions and a password.

1

Getting Started

Roadmap

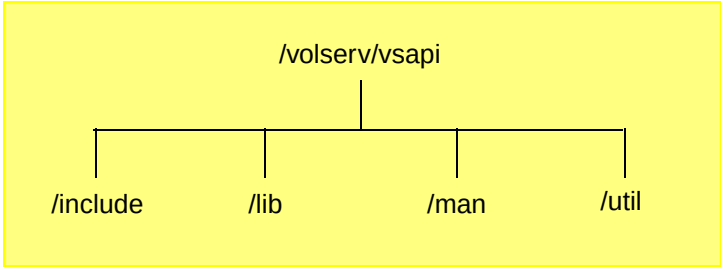
Topic	Refer To Chapter
Naming conventions. Global parameters. Error handling.	1
API functions.	2
Valid staus fields.	A
Error codes.	B
Mount example.	C

API Directory Structure

All files necessary for client interface to the VolServ software are contained in the `volserv/vsapi` directory by default. However, the installer may choose a different location during execution of the installation script, except that the `vsapi` directory is always appended to the specified location. Refer to *Installing VolServ* for more information.

Client software may use the API to interface to VolServ software. Clients are responsible for creating and linking their own applications to the appropriate API header and library files.

The default API directory structure is shown in following figure and defined in the table below.



Directory	Contents
include	Contains six header files used by API library.
lib	Contains the API library.
man	Contains man pages for all VolServ API library functions.

Directory	Contents
util	Contains various utilities used to clean up configuration problems, save copies of VolServ software logs, change the number sequence of label pattern generation, or to perform maintenance.

Application Program Interface

The VolServ Application Programmer's Interface (API) allows a programmer to interface with VolServ. The VolServ API is a set of 'C' library routines, written for a UNIX system, that the client software uses to communicate with VolServ.

The VolServ API allows the user to send commands, receive command statuses, and receive VolServ MediaClass notifications.

The VolServ API interface uses the VolServ Remote Procedure Call (RPC) interface to communicate with VolServ. All features available in the VolServ RPC interface are supported in the VolServ API.

Extensible

The VolServ API is extensible. Any change in the VolServ RPC interface is handled by the API. Thus, the client program does not have to be updated to maintain compatibility when the VolServ RPC interface is modified. The client program is updated only to use new VolServ features.

Consistent

The VolServ API is consistent. All commands and their related functions have the same interface and operate similarly.

Portable

The VolServ API is designed to be highly portable to other platforms. (If the client program is ported to a platform supported by the VolServ API, the client program does not need to be concerned with VolServ connectivity.)

Flexible

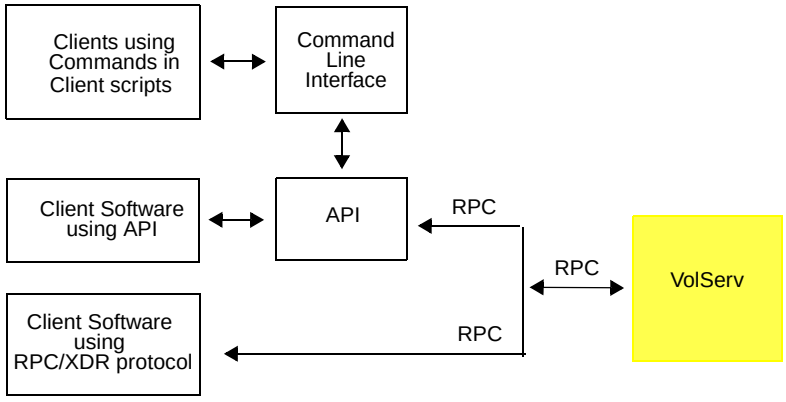
The VolServ API is flexible. A client program can send a command to VolServ and either

- Wait for final status before continuing processing (synchronous processing).
- Or, continue processing after VolServ has received the command. The client software receives the command's status at a later time (asynchronous processing).

A client program can also mix these operation modes.

Client Interface Summary

The figure below illustrates the communication paths supported by VolServ:



The following outline shows the flow of information through these communication paths.

- Clients using the Command Line Interface and Client Scripts.
  - The client-to-client script issues a request from the command line.
  - The CLI software performs first-level validation on the request and forwards the request to the API software.
  - The API software serializes the request into XDR format and transmits the request to VolServ using the RPC/XDR protocol.
  - VolServ returns data and/or status to the API software using the RPC/XDR protocol.

  - The API software deserializes the data and/or status from the XDR formats and forwards the information to the CLI software.
  - The CLI software formats the data and/or status and forwards this information to the client that issued the request.
  - The client/client script processes the data and/or status returned by the CLI software.

- Client software using the API.
  - The client software issues a request to the API software by calling an API function/routine.
  - The API software serializes the request into XDR format and transmits the request to VolServ using the RPC/XDR protocol.
  - VolServ returns data and/or status to the API software using the RPC/XDR protocol.
  - The API software deserializes the data and/or status from the XDR formats and forwards the information to the client software.
  - The client software processes the data and/or status returned by the API software.
- Client software using the RPC/XDR protocol.
  - The client software serializes the request into XDR format and transmits the request to VolServ using the RPC/XDR protocol.
  - VolServ returns data and/or status to the client software using the RPC/XDR protocol.
  - The client software deserializes the information from the XDR format and processes the returned information.

Unsolicited Communication

VolServ can generate unsolicited communication after specific client and operator commands. This unsolicited communication is referred to in some VolServ documentation as “Callbacks” or “Notifications.”

Unsolicited communication from VolServ can be directed to either:

- A preselected RPC address.
- Or, to an internet address associated with an enterprise. The RPC address or enterprise assigned for unsolicited communication is assignable at the MediaClass level. Refer to the *Command Reference* book for detailed information about defining unsolicited communication parameters for a MediaClass group.

  This address is used as the receiver for unsolicited messages that VolServ transmits. These messages are transmitted at the completion of VolServ processing that had an impact on any medium associated with the particular MediaClass group.

Running the following commands: `VSCMD_Import`, `VSCMD_Export`, `VSCMD_CheckIn`, `VSCMD_CheckOut`, `VSCMD_Mount`, `VSCMD_Dismount`, and `VSCMD_Reclassify` can generate unsolicited communication from VolServ. This unsolicited communication is returned to the client issuing the command only if that client is specified in the processing parameters as the destination for all unsolicited communication for the MediaClass group.

VolServ API Integration

To integrate the VolServ API in a client application, the client includes the header file `vs_client.h` in the source modules that reference the VolServ API types and functions. The client then links the program with the VolServ API library `libvsapi.a` with the `-lvsapi` option for the `cc` or `ld` commands.

The following five header files are delivered with the VolServ API:

Header Files	Description
vs_client.h	Includes the VolServ API header files needed by the client.
vs_defs.h	Includes the VolServ API definitions needed for the parameter lists.
vs_types.h	Includes the VolServ API types and enumerations.
vs_globals.h	Includes the VolServ API global variables that are accessible to the client.
vs_proto.h	Includes the prototypes for all VolServ API functions.

Before any command can be sent by the client program, the VolServ API must by initialized with a call to `VS_Initialize`. The routine `VS_Initialize` creates and initializes the APIs required variables and creates a communication link with VolServ. Before the client program terminates, it calls the `VS_Terminate` routine to allow the VolServ API to clean up its processing.

For example:

```
#include <stdio.h>
#include <vs_client.h>
main()
{
     VST_HOSTNAME vshost;


/* get volserv host name from the user */
printf ( “Enter the name of the VolServ host
     computer ==> “ );
scanf ( “%s”, vshost );


/* initialize the VolServ API. */
/* returns TRUE if successful, */
/* FALSE if fails. */
if ( VS_Initialize ( vshost, 0, 30 ) )
{
     /* send and create commands */
.............


     /* allow VolServ API to */
     /* terminate properly */
     VS_Terminate();
}
else
{
     printf ( “Error initializing VolServ
     API” );
     }
}
```

API Types

API objects and handles are described below.

Objects

The VolServ API uses an object-oriented metaphor for tracking VolServ items. Each object contains several fields that describe the object, routines that create and destroy the object, and routines that access the data contained within the object.

The following objects mirror items found in VolServ: archive, archive media class, archive type capacity, component, drive, drive pool, enterprise group, enterprise connection, media, MediaClass group, media type, mount selection expression, mount selection criteria, mount selection criteria group, and request. These objects usually store information relating only to their VolServ counterparts.

Note

The following objects are specific to the VolServ API: command, error, notify, status, and table. These objects contain information about the associated command and its statuses.

Handles

A handle is a pointer to a VolServ API object. An object is created by an initialization routine that returns a handle that points to the newly allocated object. After a handle is created, it is passed as a parameter to the routines that access the handle's data.

Each handle has the four base routines descried in the table below:

Routine	Description
initializer	The initializer creates and initializes the data held by the handle and returns a pointer to the client that is used as a parameter to the destructor, assignment, and accessor functions.
destructer	The destructor frees the space held by the handle.
assignment	The assignment function allows the client to assign values to fields within the handle. A call to the assignment function takes a variable argument list that includes a handle and a list of identifier-value pairs. The handle identifies the handle for which field values are being assigned. An identifier-value pair identifies the field for which a value is being assigned and the value being assigned to that field.
accessor	The accessor function allows the client to retrieve values from fields within the handle. A call to the accessor function takes a variable argument list that includes a handle and a list of identifier-pointer pairs. The handle identifies the handle for which field values are to be retrieved. An identifier-pointer pair identifies the field for which the value is to be retrieved and a pointer to the location where the field value is to be stored.

The following example creates a command handle and uses the assignment function to set the priority field within the command object to a value of 10.

```
VST_COMMAND_HANDLE cmdh;
if ( (cmdh = VS_Command_Create()) != NULL )
{
     VS_Command_SetFields (cmdh,

VSID_PRIORITY, 10,

VSID_ENDFIELD );
}
```

Naming Conventions

The table below describes the API naming conventions:

Item	Description
Handles	Handle names start with the VST prefix, followed by the specific object name, and end with the HANDLE suffix. The name should be in all capitals. For example: VST_DRIVE_HANDLE
Types	Types are defined in a client-accessed header file begin with the VST prefix. For example: VST_DRIVE_ID
Definitions	Definitions that are defined in the client definition header file should begin with the VSD prefix. For example: VSD_DRIVE_NAME_LEN
Enumerations	Enumerations have two conventions to follow. The type begins with the VST prefix. The values inside the enumeration structure begin with the VSE prefix. For example: typedef enum { VSE_DRIVETYPE_NONE = 0, VSE_DRIVETYPE_MAGTAPE =1 } VST_DRIVE_TYPE
Global Variables	Global variables begin with the VSG prefix. For example: VSG_ERROR_CODE
Identifiers	Identifiers that are used in the “assignment” and “accessor” functions to identify the field being accessed within each handle begin with the VSID prefix. The following identifier is for a drive identifier, for example: VSID_DRIVE_ID

Item	Description

Functions	Function names used in the VolServ API also follow a specific set of rules. All function names exist in the format of the VS prefix, followed by the object name, and ending with a condensed description of the function. The following example shows the overall form of function names:
	`VS_objectname_functiondescription()`
	The function descriptions also follow a naming convention, with the description coming from the following list: “GetFields”, “SetFields”, “Create”, and “Destroy.” Refer to the following example:
	`VS_Drive_Create();`
	`VS_Drive_Destroy(VS_DRIVE_HANDLE);`
	`VS_Drive_GetFields(VS_DRIVE_HANDLE, ... )`
	`VS_Drive_SetFields(VS_DRIVE_HANDLE, ... )`
	The “GetFields” and “SetFields” functions each take a variable argument list beginning with a handle. After the handle, the client specifies identifier-parameter pairs (or triples in some cases). These identifiers tell the function what field is to be accessed. The parameter tells the function either the value to be assigned to the field or the location where the filed value is to be retrieved. The identifier `VSID_ENDFIELD` must appear at the end of the variable argument list. Refer to the following example:
	`mount_status ( VST_COMMAND_HANDLE`
	`cmdh )`
	`{`
	`VST_STATUS_HANDLE statush;`
	`VST_ERROR_HANDLE errorh`
	`VST_STATUS_CODE satcode;`
	`VST_MEDIA_ID media;`
	`VST_DRIVE_ID drive;`

Item	Description
	VS_Command_GetFields (cmdh, VSID_STATUS_HANDLE, &statush, VSID_ERROR_HANDLE &errorh, VSID_STATUS_CODE, &statcode, VSID_ENDFIELD);

	if (statcode == VSE_STATUS_OK)
	{
	VS_Status_GetFields (status,
	VSID_MEDIA_ID, media,
	VSID_DRIVE_ID, &drive,
	VSID_ENDFIELD);

	printf (“media [%s] mounted on
	[%d]”, media, drive);
	}
	else
	{
	print_error_code (errorh);
	}
	}

	Function names that map directly to a VolServ Command begin with the VSCMD prefix, followed by the command name. Refer to the following example: VSCMD_command_option The following is an actual command: VSCMD_DriveQuery (VS_COMMAND_HANDLE, …);

Item	Description
Parameter Defaults	Two levels of default settings used in the API software—global defaults and command-specific defaults. • Global defaults are initialized at startup and can be set or retrieved using the `VS_Global_SetFields()` and `VS_Global_GetFields()` calls. • Command-specific defaults are set using the `VSCMD_`*commandname*`_SetDefaults()` calls. If command-specific defaults are set for a specific command (e.g., mount), they override the global defaults for all requests for execution of that command. To override a default parameter value for a specific instance of a command request, the parameter identifier and the value to be used for the parameter can be submitted on the command request (e.g., `VSCMD_Mount()`) itself.

Dispatch Routines

Dispatch routines are functions within the client software that the API automatically calls when status messages or MediaClass callbacks are received from VolServ. The dispatch routine for MediaClass callbacks (also referred to as notifications) is described on the man page for `VS_Notify_SetFields(l)`.

Dispatch routines for status messages are set for all requests with the `VS_Global_SetFields()` call, for all requests of a given type with the `VSCMD_`*request*`_SetDefaults()` call, or as part of the call that sends the VolServ request.

Dispatch routines are prototyped as follows:

- void dispatchroutine(`VST_COMMAND_HANDLE` handle)

The dispatch routine takes one argument. This is the command handle for the request that received status.

Global Parameters

Global parameters are a group of parameters that are used by the API for all VolServ requests. Most of these are sent to VolServ, but some serve as control information for the API. The following table describe these parameters.

Global Parameter	Description
VSID_CLIENT_DISPATCH	Pointer to the dispatch function for all commands.
VSID_ENTERPRISE_ID	Identifier of the enterprise, if any, to receive intermediate and final status on every command request.
VSID_NOTIFY_DISPATCH	Pointer to the dispatch function used for notification (MediaClass callback) processing.
VSID_PRIORITY (defaults to 15)	Execution priority assigned to every command request. Values range from 1 (highest) to 32 (lowest).
VSID_RETRY_LIMIT	Number of times the API software is to retry for command status from VolServ before returning a time-out to the client software. Total length of time the API software waits for a command status from VolServ is (`VSID_RETRY_LIMIT` plus1) multiplied by the `VSID_TIMEOUT_VALUE`. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG	Status wait flag for all commands. This flag controls whether the API operates in synchronous or asynchronous mode. If its value TRUE, the API waits for status (operate in synchronous mode.) If its value is FALSE, the API operates in asynchronous mode.
VSID_TIMEOUT_VALUE	Amount of time, in seconds, VolServ waits for status before timing-out to the client software for this request.
VSID_USER_FIELD	A 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for each command. Neither the API software nor VolServ uses `USER_FIELD`.

Global Variables

The following global variables are available to any software using the VolServ API.

Global Variable	Description
VSG_Error	Global error handle. It is set if an error condition occurs in a function that does not use a command handle. This includes the utility functions, handle functions, and the functions that set request- specific defaults. See the man page for `VS_Error_GetFields` to learn how to access error codes.
VSG_Command_Received	Pointer to the command handle whose status was just received from VolServ. The `VSG_Command_Received` can only be used in asynchronous processing.

API Error Handling

The API differentiates between:

- Errors that occur during API processing.
- Errors that occur within VolServ.

To accomplish this, there are two identifying parts for each error—the first part identifies where the error occurred—the second part identifies the specific error.

The API tracks errors by using error handles. There is an error handle associated with each command, as well as a global error handle. The global error handle is used to track errors that cannot be associated directly with a command (e.g., bad handle type and null handle).

The error code returned by the API is of the form: AAANNN, where AAA is a three-letter string designating where the error originates, and NNN is a numeric code designating the specific error. The three-letter string maps either to an API object or to VolServ. The numeric code represents either an API internal error or the VolServ error code returned in the command's status.

The following example shows how to access an error code:

```
int numcode,                  objcode;
VST_ERROR_CODE                errcode;
VST_ERROR_HANDLE              errorhandle;

     VS_Error_GetFields     (errorhandle,
         VSID_ERROR_OBJECT,
       &objcode,
               VSID_ERROR_NUMBER,
       &numcode,

         VSID_ERROR_CODE,
       errcode
```

```
        VSID_ENDFIELD);


printf ( “error code %s\n”, errcode );
printf ( “object code for error: %d\n”,
         objcode );
printf ( “numeric code for error: %d\n”,
         numcode );
```

Using the API

Command Handle

To send a command to VolServ, the client has to create a command handle in which the request and its status is kept. The command handle holds the information needed for the VolServ API to process the command. The information in the command handle is general for all commands (e.g., priority, time-out values). Most fields are defaulted to values that can be overridden by the client.

Command Calls

Each command has one entry point - a call of the form: `VSCMD_`*commandname*

Each command accepts the command handle and a variable length argument list with parameter name and value pairs. Each parameter name and value pair specifies a command option and its corresponding value.

The options are divided into two parts: the general command options and the command-specific options. The general command options are fields contained in all requests (e.g., priority). The specific command options are fields particular to that type of request (e.g., archive name for the archive vary command). A specific command option may not be unique to one command, it can be used in several different commands. Each command call returns a boolean value that indicates its success or failure.

See the following example:

```
if ( VSCMD_Mount ( cmd_handle,

     VSID_MEDIACLASS_NAME,”scratch”
          VSID_DRIVEPOOL_NAME,
     ”stagepool”,
          VSID_PRIORITY,1,
          VSID_ENDFIELD ) )
{
printf ( “mount successful\n” );
}
```


Command Defaults

Options that are not passed in the argument list are defaulted to command-specific defaults. These defaults are kept on a command basis and can be set to client-desired values. The function to set command-specific defaults is of the form:

`VSCMD_`*commandname*`_SetDefaults`

The defaults are specified in a variable length argument list with parameter name value pairs. The values in the command parameter list supersede global default values. See the following example:

```
VSCMD_Mount_SetDefaults (
     VSID_PRIORITY, 1,
     VSID_MEDIACLASS_NAME,”scratch”
     VSID_DRIVEPOOL_NAME,
     ”defaultpool”,
     VSID_ENDFIELD);


VSCMD_Mount (cmd_handle,


     VSID_DRIVEPOOL_NAME,”stagepool”,
          VSID_ENDFIELD );
```

These two commands perform the same function as the previous example. The `VSCMD_Mount_SetDefaults` function sets command-specific defaults for all future Mount commands within this process. When the Mount command is issued, the parameters that are not specified are defaulted to the current command-specific defaults. The defaults specified by `VSCMD_Mount_SetDefaults` are valid only for Mount commands.

Command Status

After each status received from VolServ, the pertinent status fields are stored in a status handle within the command handle. The client can get the status handle from the command handle with the `VS_Command_GetFields` function. After the status handle is obtained, any command-specific field associated with the status can be obtained through the `VS_Status_GetFields` function. See to the following example:

```
VST_MEDIA_ID     mediaid;
VST_DRIVE_ID     driveid;
VST_STATUS     status;
VST_COMMAND_HANDLE     cmd_handle;
VST_STATUS_HANDLE     status_handle;

if ( (cmd_handle = VS_Command_Create ())
     ==
          (VST_COMMAND_HANDLE) NULL )
{
     return ( -1 );
}
```

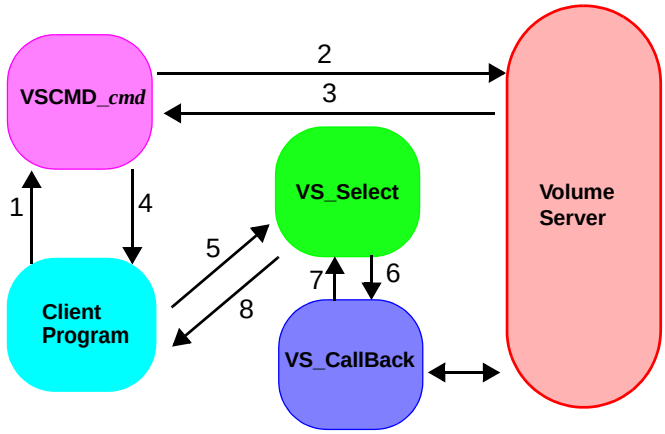
```
if ( VSCMD_Mount (cmd_handle,
          VSID_MEDIACLASS_NAME,
     ”scratch”,
          VSID_DRIVEPOOL_NAME,
     ”stagepool”
          VSID_ENDFIELD ) )
{
     VS_Command_GetFields     (
      cmd_handle,
          VSID_STATUS,     &status,
      VSID_STATUS_HANDLE,&status_handle,
          VSID_ENDFIELD             );

if ( status == VSE_STATUS_OK )

{
     VS_Status_GetFields (
     status_handle,
          VSID_MEDIA_ID,
     mediaid,
          VSID_DRIVE_ID,
&driveid,
          VSID_ENDFIELD );


printf ( “Media [%s] mounted on
drive,
     [%d]\n”,mediaid, driveid );
     }
}
```

Processing

The VolServ API allows for both asynchronous and synchronous processing of VolServ commands.

Asynchronous

For asynchronous processing, the VolServ API returns control to the client after initial status is received.



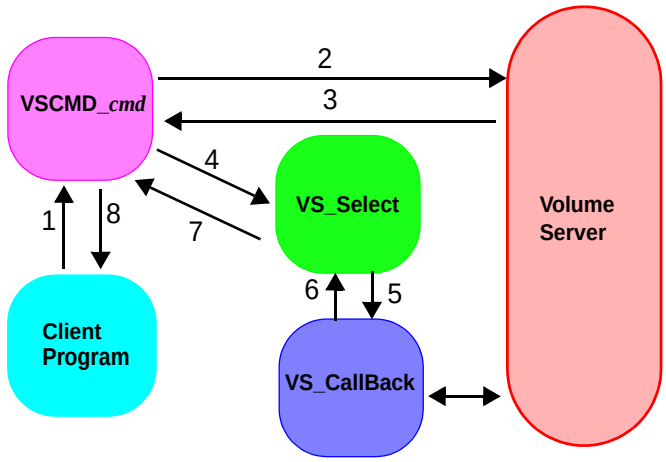
1. Client Program calls the command function (VSCMD_*cmd*).
2. The VSCMD_*cmd* calls the Volume Server.
3. VolServ returns initial status to VSCMD_*cmd*.
4. VSCMD_*cmd* returns control to the Client Program.
5. The Client Program calls the VS_Select loop to wait for status.
6. VS_Select calls the command specific VS_CallBack.
7. VS_CallBack returns status to the VS_Select.
8. VS_Select returns status to the Client Program.

To receive subsequent status from VolServ API, the client invokes the VolServ APIs `VS_Select` function. It is the responsibility of the client to place the VolServ API into its select loop so all subsequent statuses can be received. In asynchronous processing, the client can issue multiple VolServ commands and immediately receive their statuses.

Asynchronous processing also gives the client more control over the processing environment. The `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode. If the `VSID_STATUS_WAIT_FLAG` value is FALSE, the API operates in asynchronous mode.

Synchronous

For synchronous processing, the VolServ API returns control to the client only after final status (or time-out) is received.



1. Client Program calls the command function (VSCMD_*cmd*).
2. The VSCMD_*cmd* calls the Volume Server.
3. VolServ returns initial status to VSCMD_*cmd*.
4. VSCMD_*cmd* calls the VS_Select loop to wait for status.
5. VS_Select calls the command specific VS_Callback.
6. VS_Callback returns status to VS_Select.
7. VS_Select returns control to VSCMD_*cmd*.
8. VSCMD_*cmd* returns control to the Client Program.

Synchronous processing allows processing of only one command at a time. If the `VSID_STATUS_WAIT_FLAG` value is TRUE, the API operates in synchronous mode.

API Functions

The API function descriptions in this book are presented as follows:

Function	Description
vsapi	Provides a general introduction to VolServ API processing.
VS_*cmdname*	Describes functions used to create, destroy, assign, and access handles. These functions are listed in alphabetical within the subgroup.
VSCMD_*cmdname*	Describes how a user accesses VS commands through the API. These functions are listed in alphabetical order within the subgroup. The VolServ API allows for both asynchronous and synchronous processing of VolServ commands.

NOTES

NOTES

2

API Functions

Roadmap

Topic	Refer To Chapter
Naming conventions. Global parameters. Error handling.	1
API functions.	2
Valid staus fields.	A
Error codes.	B
Mount example.	C

VS_Archive_Create

`VS_Archive_Create` allocates a VolServ API archive handle. An archive handle is used to pass archive information to and from VolServ.

Synopsis

VST_ARCHIVE_HANDLE VS_Archive_Create (void)

Arguments

None

Return Values

`VS_Archive_Create` returns:

- An archive handle, if one can be allocated.
- NULL, if an archive handle cannot be allocated. An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_archive_handle
4  *
5  * PURPOSE:
6  * This function tests an archive handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_archive_handle(void)
14 #else
15    VST_BOOLEAN vst_archive_handle(void)
16 #endif
17 {
```

```
18    VST_BOOLEAN                rc =
         VSE_FALSE;
19    VST_ARCHIVE_HANDLE         h;
20    VST_ARCHIVE_TYPE
         ArchiveType;
21    VST_ARCHIVE_NAME
         ArchiveName;
22    VST_HOSTNAME
         ConsoleLoc;
23    VST_COMP_STATE
         ComponentState;
24    VST_ARCHIVE_MODE
         ArchiveMode;
25    VST_ARCHIVE_FILL_MODE      FillMode;
26    VST_ARCHIVE_CONFIG_STATE
         ConfigState;
27
28    /* create the handle */
29    h = VS_Archive_Create();
30    if (h != (VST_ARCHIVE_HANDLE) NULL)
31    {
32       /* get values from user */
33       printf("*** Archive Handle
         ***\n");
34       printf("Enter Archive Type ==> ");
35       ArchiveType = atoi(gets(input));;
36       printf("Enter Archive Name ==> ");
37       gets(ArchiveName);
38       printf("Enter Console Display
         Location ==> ");
39       gets(ConsoleLoc);
40       printf("Enter archive state ==>
         ");
41       ComponentState =
         atoi(gets(input));
42       printf("Enter Archive Mode ==> ");
43       ArchiveMode = atoi(gets(input));
44       printf("Enter Fill Mode ==> ");
45       FillMode = atoi(gets(input));;
46       printf("Enter Config State ==> ");
47       ConfigState = atoi(gets(input));;
48       /* set the fields in the handle */
```

```
49       rc = VS_Archive_SetFields(h,
50          VSID_ARCHIVE_NAME,
         ArchiveName,
51          VSID_ARCHIVE_TYPE,
         ArchiveType,
52
         VSID_ARCHIVE_CONSOLE_LOCATION,Con
         soleLoc,
53          VSID_COMP_STATE,
         ComponentState,
54          VSID_ARCHIVE_MODE,
         ArchiveMode,
55          VSID_ARCHIVE_FILL_MODE,
         FillMode,
56          VSID_ARCHIVE_CONFIG_STATE,
         ConfigState,
57          VSID_ENDFIELD);
58       if (rc)
59       {
60          vst_print_archive(h);
61       }
62       VS_Archive_Destroy(h);
63    }
64 return(rc);
65 }
```

See Also

- vsapi(l),
- VS_Archive_Destroy(l),
- VS_Archive_GetFields(l),
- VS_Archive_SetFields(l),
- VS_Error_GetFields(l)

VS_Archive_Destroy

VS_Archive_Destroy deallocates an archive handle that was allocated with VS_Archive_Create. An archive handle is used to pass archive information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Archive_Destroy
(VST_ARCHIVE_HANDLE handle)

Arguments

- handle = Archive handle to destroy.

Return Values

- VS_Archive_Destroy returns:
- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not an archive handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
66 /*************************************
       **********
67 * FUNCTION: vst_archive_handle
68 *
69 * PURPOSE:
70 * This function tests an archive handle.
71 * PARAMETERS:
72 * none
73 *
74 *************************************
       *********/
75 #ifdef ANSI_C
76    VST_BOOLEAN vst_archive_handle(void)
77 #else
```

```
78    VST_BOOLEAN vst_archive_handle(void)
79 #endif
80 {
81    VST_BOOLEAN                rc =
         VSE_FALSE;
82    VST_ARCHIVE_HANDLE         h;
83    VST_ARCHIVE_TYPE
         ArchiveType;
84    VST_ARCHIVE_NAME
         ArchiveName;
85    VST_HOSTNAME
         ConsoleLoc;
86    VST_COMP_STATE
         ComponentState;
87    VST_ARCHIVE_MODE
         ArchiveMode;
88    VST_ARCHIVE_FILL_MODE      FillMode;
89    VST_ARCHIVE_CONFIG_STATE
         ConfigState;
90
91    /* create the handle */
92    h = VS_Archive_Create();
93    if (h != (VST_ARCHIVE_HANDLE) NULL)
94    {
95       /* get values from user */
96       printf("*** Archive Handle
         ***\n");
97       printf("Enter Archive Type ==> ");
98       ArchiveType = atoi(gets(input));;
99       printf("Enter Archive Name ==> ");
100      gets(ArchiveName);
101      printf("Enter Console Display
         Location ==> ");
102      gets(ConsoleLoc);
103      printf("Enter archive state ==>
         ");
104      ComponentState =
         atoi(gets(input));
105      printf("Enter Archive Mode ==> ");
106      ArchiveMode = atoi(gets(input));
107      printf("Enter Fill Mode ==> ");
108      FillMode = atoi(gets(input));;
```

```
109      printf("Enter Config State ==> ");
110      ConfigState = atoi(gets(input));;
111      /* set the fields in the handle */
112      rc = VS_Archive_SetFields(h,
113         VSID_ARCHIVE_NAME,
         ArchiveName,
114         VSID_ARCHIVE_TYPE,
         ArchiveType,
115
         VSID_ARCHIVE_CONSOLE_LOCATION,Con
         soleLoc,
116         VSID_COMP_STATE,
         ComponentState,
117         VSID_ARCHIVE_MODE,
         ArchiveMode,
118         VSID_ARCHIVE_FILL_MODE,
         FillMode,
119         VSID_ARCHIVE_CONFIG_STATE,
         ConfigState,
120         VSID_ENDFIELD);
121      if (rc)
122      {
123         vst_print_archive(h);
124      }
125      VS_Archive_Destroy(h);
126   }
127return(rc);
128}
```


Notes

After `VS_Archive_Destroy` has been called for an archive handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Archive_Create(l),
- VS_Archive_GetFields(l),
- VS_Archive_SetFields(l),
- VS_Error_GetFields(l)

VS_Archive_GetFields

`VS_Archive_GetFields` retrieves information associated with an archive handle. An archive handle is used to pass archive information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Archive_GetFields
(VST_ARCHIVE_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Archive handle where information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_CONFIG_STATE (VST_ARCHIVE_CONFIG_STATE*)	Pointer to a boolean flag that indicates whether the archive is currently being configured or reconfigured. Valid `VSID_ARCHIVE_CONFG_STATE` values are enumerated in the *vs_types.h* file.
VSID_ARCHIVE_CONSOLE_LOCATION (VST_HOSTNAME)	Pointer to the location of the archive's console display.

Parameter Type	Description
VSID_ARCHIVE_FILL_MODE (VST_ARCHIVE_FILL_MODE*)	Pointer to the method of allocating bins to new media as they are entered into an archive. `VSID_ARCHIVE_FILL_MODE` is applicable only to the DataShelf and DataLibrary archives. Valid `VSID_ARCHIVE_FILL_MODE` values are enumerated in the *vs_types.h* file.
VSID_ARCHIVE_MODE (VST_ARCHIVE_MODE *)	Pointer that specifies whether this archive is attended by an operator to handle media movement commands that require human intervention. Valid `VSID_ARCHIVE_MODE` values are enumerated in the *vs_types.h* file.
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Pointer to the name of this archive. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ARCHIVE_TYPE (VST_ARCHIVE_TYPE *)	Pointer to the type of this archive. Valid `VSID_ARCHIVE_TYPE` values are enumerated in the *vs_types.h* file.
VSID_ARCHIVEMEDIACLASS_HANDLE (int)	Index of the archive media class handle in the MediaClass capacity table.
(VST_ARCHIVEMEDIACLASS_HANDLE *)	Pointer to the first archive media class handle in the MediaClass capacity table.
VSID_ARCHIVEMEDIACLASS_HANDLE_ENTRY (int,VST_ARCHIVEMEDIACLASS_HANDLE)	Index of the archive media class handle in the MediaClass capacity table. Pointer to the location to store the archive media class handle.
VSID_ARCHIVEMEDIACLASS_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the MediaClass capacity (in table format) for this archive.
VSID_COMP_STATE (VST_COMP_STATE *)	Pointer to the operational state of this archive. Valid `VSID_COMP_STATE` values are enumerated in the *vs_types.h* file.
VSID_DRIVE_ID (int)	Index of the drive in the drive identifier table.

Parameter Type	Description
(VST_DRIVE_ID *)	Pointer to the first drive id in the drive identifier table.
VSID_DRIVE_ID_ENTRY (int, VST_Drive_ID *)	Index of the drive in the drive identifier table. Pointer to the location to store the drive identifier
VSID_DRIVE_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the drives (in table format) associated with this archive.
VSID_MEDIA_ID (int)	Index of the medium in the media identifier table.
(VST_MEDIA_ID)	Pointer to the first media id in the media identifier table.
VSID_MEDIA_ID_ENTRY (int, VST_ MEDIA_ID)	Index of the medium in the media identifier table. Pointer to the location to store the media identifier.
VSID_MEDIA_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the media identifiers (in table format) that are currently in this archive.
VSID_NUMBER_ARCHIVEMEDIACLASS_HANDLES (int *)	Pointer to the number of archive media class handles present in the archive media class handle table.
VSID_NUMBER_DRIVE_ID (int *)	Pointer to the number of drive ids present in the drive id table.
VSID_NUMBER_MEDIA_IDS (int *)	Pointer to the number of media ids present in the media id table.
VSID_NUMBER_TYPECAPACITY_HANDLES (int *)	Pointer to the number of type capacity handles present in the type capacity handle table.
VSID_TYPECAPACITY_HANDLE (int)	Index of the type capacity handle in the table.
(VST_TYPECAPACITY_HANDLE *)	Pointer to the first archive type capacity handle in the MediaType capacity table.

Parameter Type	Description
VSID_TYPECAPACITY_HANDLE_ENTRY (int, VSID_TYPECAPACITY_HANDLE *)	Index of the type capacity handle in the MediaType capacity table. Pointer to the location to store the type capacity handle.
VSID_TYPECAPACITY_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the type capacity (in table format) for this archive.

Return Values

`VS_Archive_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not an archive handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_OUTOFRANGE` - An index value was out of range.

Example

```
129/*************************************
        **********
130*
131* FUNCTION: vst_print_archive
132*
133* PURPOSE:
134* This function prints out the
        information stored in
135* an archive handle.
136*
137* PARAMETERS:
138* h : the archive handle to print
139*
140*************************************
        **********/
141#ifdef ANSI_C
142   void
        vst_print_archive(VST_ARCHIVE_HAN
        DLE h)
143#else
144   void vst_print_archive(h)
145   VST_ARCHIVE_HANDLE h;
146#endif
147{
148   VST_ARCHIVE_TYPE
        ArchiveType;
149   VST_ARCHIVE_NAME
        ArchiveName;
150   VST_HOSTNAME
        ConsoleLoc;
151   VST_COMP_STATE
        ComponentState;
152   VST_ARCHIVE_MODE
        ArchiveMode;
153   VST_ARCHIVE_FILL_MODE      FillMode;
154   VST_ARCHIVE_CONFIG_STATE
        ConfigState;
155   VST_TABLE_HANDLE
        DriveIDTable;
156   VST_TABLE_HANDLE
        MediaIDTable;
```

```
157   VST_TABLE_HANDLE
         ClassCapacityTable;
158   VST_TABLE_HANDLE
         TypeCapacityHandleTable;
159   int                              n;
160   int                              i;
161   VST_DRIVE_ID *
         DriveID;
162   char *
         MediaID;
163   VST_ARCHIVEMEDIACLASS_HANDLE
         arcmc_handle;
164   VST_TYPECAPACITY_HANDLE
         typecap_handle;
165
166   VS_Archive_GetFields(h,
167      VSID_ARCHIVE_NAME,
         ArchiveName,
168      VSID_ARCHIVE_TYPE,
         &ArchiveType,
169      VSID_ARCHIVE_CONSOLE_LOCATION,
         ConsoleLoc,
170      VSID_COMP_STATE,
         &ComponentState,
171      VSID_ARCHIVE_MODE,
         &ArchiveMode,
172      VSID_ARCHIVE_FILL_MODE,
         &FillMode,
173      VSID_ARCHIVE_CONFIG_STATE,
         &ConfigState,
174      VSID_DRIVE_ID_TABLE,
         &DriveIDTable,
175      VSID_MEDIA_ID_TABLE,
         &MediaIDTable,
176
         VSID_ARCHIVEMEDIACLASS_HANDLE_TAB
         LE, &ClassCapacityTable,
177      VSID_TYPECAPACITY_HANDLE_TABLE,
         &TypeCapacityHandleTable,
178      VSID_ENDFIELD);
179
```

```
180   printf("*** Archive Handle
         Information ***\n");
181   printf("Archive Type = %d\n",
         ArchiveType);
182   printf("Archive Name = %s\n",
         ArchiveName);
183   printf("Console Display Location =
         %s\n", ConsoleLoc);
184   printf("Component State = %d\n",
         ComponentState);
185   printf("Archive Mode = %d\n",
         ArchiveMode);
186   printf("Archive Fill Mode = %d\n",
         FillMode);
187   printf("Config State = %d\n",
         ConfigState);
188
189   if (DriveIDTable !=
         (VST_TABLE_HANDLE) NULL)
190   {
191      /* Get # of entries */
192      VS_Table_GetFields(DriveIDTable,
193
         VSID_NUMBER_ENTRIES, &n,
194                     VSID_ENDFIELD);
195      for ( i = 0; i < n; i++)
196      {
197
         VS_Table_GetFields(DriveIDTable,
198                     VSID_TABLE_ENTRY,
         i, &DriveID,
199                     VSID_ENDFIELD);
200      printf("DriveID Entry #%d =
         %d\n",i,*DriveID);
201      }
202   }
203
204   if (MediaIDTable !=
         (VST_TABLE_HANDLE) NULL)
205   {
206      /* Get # of entries */
207      VS_Table_GetFields(MediaIDTable,
```

```
208
        VSID_NUMBER_ENTRIES, &n,
209                     VSID_ENDFIELD);
210     for ( i = 0; i < n; i++)
211     {
212
        VS_Table_GetFields(MediaIDTable,
213                     VSID_TABLE_ENTRY,
        i, &MediaID,
214                     VSID_ENDFIELD);
215     printf("Media ID Entry #%d =
        %s\n",i,MediaID);
216     }
217  }
218
219  if (ClassCapacityTable !=
        (VST_TABLE_HANDLE) NULL)
220  {
221     /* Get # of entries */
222
        VS_Table_GetFields(ClassCapacityT
        able,
223
        VSID_NUMBER_ENTRIES, &n,
224                     VSID_ENDFIELD);
225     for ( i = 0; i < n; i++)
226     {
227
        VS_Table_GetFields(ClassCapacityT
        able,
228           VSID_TABLE_ENTRY, i,
        &arcmc_handle,
229           VSID_ENDFIELD);
230
        vst_print_archivemediaclass(arcmc
        _handle);
231     }
232  }
233
234  if (TypeCapacityHandleTable !=
        (VST_TABLE_HANDLE) NULL)
235  {
```

```
236       /* Get # of entries */
237
          VS_Table_GetFields(TypeCapacityHa
          ndleTable,
238
          VSID_NUMBER_ENTRIES, &n,
239                       VSID_ENDFIELD);
240       for ( i = 0; i < n; i++)
241       {
242
          VS_Table_GetFields(TypeCapacityHa
          ndleTable,
243             VSID_TABLE_ENTRY, i,
          &typecap_handle,
244             VSID_ENDFIELD);
245
          vst_print_typecapacity(typecap_ha
          ndle);
246       }
247    }
248}
```


Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

VolServ assigns additional bins according to one of two user-specified algorithms: “wrap” or “first fill.” Using the wrap algorithm, VolServ assigns additional bins in order until the last bin in the archive has been assigned. VolServ then wraps to the first physical bin, goes through the bins in order, and assigns empty bins as they are encountered. Using the first fill algorithm, VolServ starts looking for an available bin at the first physical bin location. The first empty, on-line bin encountered is assigned.

The VST_ARCHIVEMEDIACLASS_HANDLE, VST_DRIVE_ID, VST_MEDIA_ID, and VST_TYPECAPACITY_HANDLE parameters require that two arguments be passed instead of one.

- The first argument passed is the entry number in the appropriate table.
- The second argument is Pointer to the location where the value is stored.

See Also

- vsapi(l),
- VS_Archive_Create(l),
- VS_Archive_Destroy(l),
- VS_Archive_SetFields(l),
- VS_ArchiveMediaClass_GetFields(l),
- VS_ArchiveMediaClass_SetFields(l),
- VS_Error_GetFields(l),
- VS_Table_GetFields(l),
- VS_TypeCapacity_GetFields(l),
- VS_TypeCapacity_SetFields(l),
- VSCMD_Archive_Query(l

VS_Archive_SetFields

`VS_Archive_SetFields` sets the value of one or more field in an archive handle. An archive handle is used to pass archive information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Archive_SetFields
(VST_ARCHIVE_HANDLE
handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Archive handle where information is stored.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD = Re`quired at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_CONFIG_STATE (VST_ARCHIVE_CONFIG_STATE)	A boolean flag that indicates whether the archive is currently being configured or reconfigured. Valid `VSID_ARCHIVE_CONFIG_STATE` values are enumerated in the *vs_types.h* file.
VSID_ARCHIVE_CONSOLE_LOCATION (VST_HOSTNAME)	The location of the archive's console display.

Parameter Type	Description
VSID_ARCHIVE_FILL_MODE (VST_ARCHIVE_FILL_MODE)	The method of allocating bins to new media as they are entered into an archive. `VSID_ARCHIVE_FILL_MODE` is applicable only to the DataShelf and DataLibrary archives. Valid `VSID_ARCHIVE_FILL_MODE` values are enumerated in the *vs_types.h* file.
VSID_ARCHIVEMEDIACLASS_HANDLE_TABLE (VST_TABLE_HANDLE)	The MediaClass capacity (in table format) for this archive.
VSID_ARCHIVE_MODE (VST_ARCHIVE_MODE)	Specifies whether this archive is attended by an operator to handle media movement commands that require human intervention. Valid `VSID_ARCHIVE_MODE` values are enumerated in the *vs_types.h* file.
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The name of this archive. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ARCHIVE_TYPE (VST_ARCHIVE_TYPE)	Type of this archive. Valid `VSID_ARCHIVE_TYPE` values are enumerated in the *vs_types.h* file.
VSID_COMP_STATE (VST_COMP_STATE)	The operational state of this archive. Valid `VSID_COMP_STATE` values are enumerated in the *vs_types.h* file.
VSID_DRIVE_ID_TABLE (VST_TABLE_HANDLE)	The drive identifiers (in table format) associated with this archive.
VSID_MEDIA_ID_TABLE (VST_TABLE_HANDLE)	The media identifiers (in table format) that are currently in this archive.
VSID_TYPECAPACITY_HANDLE_TABLE (VST_TABLE_HANDLE)	The type capacity (in table format) for this archive.

Return Values

`VS_Archive_SetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADHANDLE` - Specified handle was not a criteria handle.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
249/*************************************
        ***********
250*
251* FUNCTION: vst_archive_handle
252*
253* PURPOSE:
254* This function tests an archive handle.
255*
256* PARAMETERS:
257* none
258*
259*************************************
        **********/
260#ifdef ANSI_C
261   VST_BOOLEAN vst_archive_handle(void)
262#else
263   VST_BOOLEAN vst_archive_handle(void)
264#endif
265{
266   VST_BOOLEAN                rc =
        VSE_FALSE;
```

```
267   VST_ARCHIVE_HANDLE         h;
268   VST_ARCHIVE_TYPE
         ArchiveType;
269   VST_ARCHIVE_NAME
         ArchiveName;
270   VST_HOSTNAME
         ConsoleLoc;
271   VST_COMP_STATE
         ComponentState;
272   VST_ARCHIVE_MODE
         ArchiveMode;
273   VST_ARCHIVE_FILL_MODE      FillMode;
274   VST_ARCHIVE_CONFIG_STATE
         ConfigState;
275
276   /* create the handle */
277   h = VS_Archive_Create();
278   if (h != (VST_ARCHIVE_HANDLE) NULL)
279   {
280      /* get values from user */
281      printf("*** Archive Handle
         ***\n");
282      printf("Enter Archive Type ==> ");
283      ArchiveType = atoi(gets(input));;
284      printf("Enter Archive Name ==> ");
285      gets(ArchiveName);
286      printf("Enter Console Display
         Location ==> ");
287      gets(ConsoleLoc);
288      printf("Enter archive state ==>
         ");
289      ComponentState =
         atoi(gets(input));
290      printf("Enter Archive Mode ==> ");
291      ArchiveMode = atoi(gets(input));
292      printf("Enter Fill Mode ==> ");
293      FillMode = atoi(gets(input));;
294      printf("Enter Config State ==> ");
295      ConfigState = atoi(gets(input));;
296      /* set the fields in the handle */
297      rc = VS_Archive_SetFields(h,
```

```
298         VSID_ARCHIVE_NAME,
          ArchiveName,
299         VSID_ARCHIVE_TYPE,
          ArchiveType,
300         VSID_ARCHIVE_CONSOLE_LOCATION,
          ConsoleLoc,
301         VSID_COMP_STATE,
          ComponentState,
302         VSID_ARCHIVE_MODE,
          ArchiveMode,
303         VSID_ARCHIVE_FILL_MODE,
          FillMode,
304         VSID_ARCHIVE_CONFIG_STATE,
          ConfigState,
305         VSID_ENDFIELD);
306       if (rc)
307       {
308         vst_print_archive(h);
309       }
310       VS_Archive_Destroy(h);
311    }
312return(rc);
313}
```


Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

VolServ assigns additional bins according to one of two user-specified algorithms: “wrap” or “first fill.” Using the wrap algorithm, VolServ assigns additional bins in order until the last bin in the archive has been assigned. VolServ then wraps to the first physical bin, goes through the bins in order, and assigns empty bins as they are encountered. Using the first fill algorithm, VolServ starts looking for an available bin at the first physical bin location. The first empty, on-line bin encountered is assigned.

See Also

- vsapi(l),
- VS_Archive_Create(l),
- VS_Archive_Destroy(l),
- VS_Archive_SetFields(l),
- VS_ArchiveMediaClass_GetFields(l),
- VS_ArchiveMediaClass_SetFields(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VS_Table_Create(l),
- VS_Table_Destroy(l),
- VS_Table_GeFields(l),
- VS_Table_SetFields(l),
- VS_TypeCapacity_GetFields(l),
- VS_TypeCapacity_SetFields(l)

VS_ArchiveMediaClass_Create

`VS_ArchiveMediaClass_Create` allocates a VolServ API archive media class handle. An archive media class handle is used to pass archive media class information to and from VolServ.

Synopsis

VST_ARCHIVEMEDIACLASS_HANDLE
VS_ArchiveMediaClass_Create
(void)

Arguments

None

Return Values

`VS_ArchiveMediaClass_Create` returns:

- An archive media class handle, if one can be allocated.
- NULL, if an archive media class handle cannot be allocated. An appropriate error code is set in `VSG_Error.`
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
314/************************************
        ***********
315*
316* FUNCTION:
        vst_archivemediaclass_handle
317*
318* PURPOSE:
319* This function tests an
        archivemediaclass handle.
320*
321* PARAMETERS:
322* none
323*
324************************************
        **********/
```

```
#ifdef ANSI_C
   VST_BOOLEAN
      vst_archivemediaclass_handle(void
      )
#else
   VST_BOOLEAN
      vst_archivemediaclass_handle()
#endif
{
   VST_BOOLEAN rc;
   VST_ARCHIVEMEDIACLASS_HANDLE  h;
   VST_ARCHIVE_NAME
      Archive;
   VST_MEDIA_CLASS_NAME
      MediaClass;
   VST_MEDIA_TYPE_NAME
      MediaType;
   VST_CAPACITY
      Capacity;
   VST_PERCENT
      MediaClassPercent;
   VST_ARCHIVE_ACTION_OPTION
      ActionMode;
   VST_HIGH_MARK
      HighMark;
   VST_LOW_MARK
      LowMark;
   VST_FILL_LEVEL
      FillLevel;
   VST_PRIORITY
      MigrationPriority;
   VST_ARCHIVE_NAME
      TargetArchive;

   /* create the handle */
   h = VS_ArchiveMediaClass_Create();
   if (h !=
      (VST_ARCHIVEMEDIACLASS_HANDLE)
      NULL)
   {
      /* get the values from the user */
```

```
350      printf("** Archive Media Class
         handle **\n");
351      printf("Enter Archive Name ==> ");
352      gets(Archive);
353      printf("Enter Media Class Name ==>
         ");
354      gets(MediaClass);
355      printf("Enter Media Type Name ==>
         ");
356      gets(MediaType);
357      printf("Enter Capacity ==> ");
358      Capacity = atoi(gets(input));
359      printf("Enter MediaClass Percent
         ==> ");
360      MediaClassPercent =
         atoi(gets(input));
361      printf("Enter Archive Action Mode
         ==> ");
362      ActionMode = atoi(gets(input));
363      printf("Enter High Mark Mode ==>
         ");
364      HighMark = atoi(gets(input));
365      printf("Enter Fill Level Mode ==>
         ");
366      FillLevel = atoi(gets(input));
367      printf("Enter Migration Priority
         ==> ");
368      MigrationPriority =
         atoi(gets(input));
369      printf("Enter Target Archive ==>
         ");
370      gets(TargetArchive);
371      rc =
         VS_ArchiveMediaClass_SetFields(h,
372         VSID_ARCHIVE_NAME,
         Archive,
373         VSID_MEDIA_CLASS_NAME,
         MediaClass,
374         VSID_MEDIA_TYPE_NAME,
         MediaType,
375         VSID_CAPACITY,
         Capacity,
```

```
376         VSID_PERCENT,
          MediaClassPercent,
377         VSID_ARCHIVE_ACTION,
          ActionMode,
378         VSID_HIGH_MARK,
          HighMark,
379         VSID_LOW_MARK,
          LowMark,
380         VSID_FILL_LEVEL,
          FillLevel,
381
          VSID_MIGRATION_PRIORITY,Migration
          Priority,
382
          VSID_TARGET_ARCHIVE_NAME,TargetAr
          chive,
383         VSID_ENDFIELD);
384       if (rc)
385       {
386
          vst_print_archivemediaclass(h);
387       }
388       VS_ArchiveMediaClass_Destroy(h);
389    }
390    return(rc);
391}
```


Notes

None

See Also

- vsapi(l),
- VS_ArchiveMediaClass_Destroy(l)
- VS_ArchiveMediaClass_GetFields(l),
- VS_ArchiveMediaClass_SetFields(l),
- VS_Error_GetFields(l)

VS_ArchiveMediaClass_Destroy

`VS_ArchiveMediaClass_Destroy` deallocates an archive media class handle that was allocated with `VS_ArchiveMediaClass_Create`.

Synopsis

VST_BOOLEAN VS_ArchiveMediaClass_Destroy
(VST_ARCHIVEMEDIACLASS
_HANDLE
handle)

Arguments

- `handle` = Archive media class handle to be destroyed.

Return Values

`VS_ArchiveMediaClass_Destroy` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADHANDLE` - Specified handle was not an archive media class handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_archivemediaclass_handle
4  *
5  * PURPOSE:
6  * This function tests an
       archivemediaclass handle.
7  *
```

```
8  * PARAMETERS:
9  * none
10 *
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_archivemediaclass_handle(void
         )
14 #else
15    VST_BOOLEAN
         vst_archivemediaclass_handle()
16 #endif
17 {
18    VST_BOOLEAN rc;
19    VST_ARCHIVEMEDIACLASS_HANDLE  h;
20    VST_ARCHIVE_NAME
         Archive;
21    VST_MEDIA_CLASS_NAME
         MediaClass;
22    VST_MEDIA_TYPE_NAME
         MediaType;
23    VST_CAPACITY
         Capacity;
24    VST_PERCENT
         MediaClassPercent;
25    VST_ARCHIVE_ACTION_OPTION
         ActionMode;
26    VST_HIGH_MARK
         HighMark;
27    VST_LOW_MARK
         LowMark;
28    VST_FILL_LEVEL
         FillLevel;
29    VST_PRIORITY
         MigrationPriority;
30    VST_ARCHIVE_NAME
         TargetArchive;
31
32    /* create the handle */
33    h = VS_ArchiveMediaClass_Create();
```

```
34    if (h !=
         (VST_ARCHIVEMEDIACLASS_HANDLE)
         NULL)
35    {
36       /* get the values from the user */
37       printf("** Archive Media Class
         handle **\n");
38       printf("Enter Archive Name ==> ");
39       gets(Archive);
40       printf("Enter Media Class Name ==>
         ");
41       gets(MediaClass);
42       printf("Enter Media Type Name ==>
         ");
43       gets(MediaType);
44       printf("Enter Capacity ==> ");
45       Capacity = atoi(gets(input));
46       printf("Enter MediaClass Percent
         ==> ");
47       MediaClassPercent =
         atoi(gets(input));
48       printf("Enter Archive Action Mode
         ==> ");
49       ActionMode = atoi(gets(input));
50       printf("Enter High Mark Mode ==>
         ");
51       HighMark = atoi(gets(input));
52       printf("Enter Fill Level Mode ==>
         ");
53       FillLevel = atoi(gets(input));
54       printf("Enter Migration Priority
         ==> ");
55       MigrationPriority =
         atoi(gets(input));
56       printf("Enter Target Archive ==>
         ");
57       gets(TargetArchive);
58       rc =
         VS_ArchiveMediaClass_SetFields(h,
59          VSID_ARCHIVE_NAME,
         Archive,
```

```
60          VSID_MEDIA_CLASS_NAME,
          MediaClass,
61          VSID_MEDIA_TYPE_NAME,
          MediaType,
62          VSID_CAPACITY,
          Capacity,
63          VSID_PERCENT,
          MediaClassPercent,
64          VSID_ARCHIVE_ACTION,
          ActionMode,
65          VSID_HIGH_MARK,
          HighMark,
66          VSID_LOW_MARK,
          LowMark,
67          VSID_FILL_LEVEL,
          FillLevel,
68          VSID_MIGRATION_PRIORITY,
          MigrationPriority,
69          VSID_TARGET_ARCHIVE_NAME,
          TargetArchive,
70          VSID_ENDFIELD);
71        if (rc)
72        {
73
          vst_print_archivemediaclass(h);
74        }
75        VS_ArchiveMediaClass_Destroy(h);
76     }
77     return(rc);
78 }
```

Notes

After `VS_ArchiveMediaClass_Destroy` has been called for an archive media class handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_ArchiveMediaClass_Create(l),
- VS_ArchiveMediaClass_GetFields(l),
- VS_ArchiveMediaClass_SetFields(l),
- VS_Error_GetFields(l)

VS_Archive MediaClass_ GetFields

`VS_ArchiveMediaClass_GetFields` retrieves information associated with an archive media class handle. An archive media class handle is used to pass archive media class information to and from VolServ.

Synopsis

```
VST_BOOLEAN VS_ArchiveMediaClass_GetFields
( VST_ARCHIVEMEDIACLASS
_HANDLE handle,
“…”,
VSID_ENDFIELD )
```

Arguments

- `handle` =Archive media class handle where information is retrieved.
- “…” = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_ACTION (VST_ARCHIVE_ACTION_OPTION*)	Pointer to the archive action VolServ is to take when the number of media in the archive media class exceeds the specified high mark threshold. Valid `VSID_ARCHIVE_ACTION` values are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Pointer to the name of the archive associated with the archive media class relationship. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CAPACITY (VST_CAPACITY *)	Pointer to the percentage of the total MediaClass capacity that can be stored in this archive.
VSID_COMPONENT_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the preferred locations for media assigned to this archive media class.
VSID_FILL_LEVEL (VST_FILL_LEVEL *)	Pointer to the number of media currently in this archive media class.
VSID_HIGH_MARK (VST_HIGH_MARK *)	The percentage of `VSID_CAPACITY` above which the specified migration policy option is performed or initiated. This field is applicable only if `VSID_ARCHIVE_ACTION` is set to `VSE_ARCHIVE_ACTION_NOTIFY` or `VSE_ARCHIVE_ACTION_MIG`.
VSID_LOW_MARK (VST_LOW_MARK *)	Pointer to the percentage of the archive media class capacity below which automatic migration of media out of the archive media class stops. This field is applicable only if `VSID_ARCHIVE_ACTION` is set to `VSE_ARCHIVE_ACTION_NOTIFY` or `VSE_ARCHIVE_ACTION_MIG`.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Pointer to the MediaClass name associated with the archive media class relationship. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Parameter Type	Description
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	Pointer to the media type associated with the archive media class. Valid media type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MIGRATION_PRIORITY (VST_PRIORITY *)	Pointer to the migration priority to be applied to this archive media class.
VSID_NUMBER_COMPONENT_HANDLES (int *)	Pointer to the number of component handles present in the component handle table.
VSID_PERCENT (VST_PERCENT *)	Pointer to the percentage of the media assigned to the related MediaClass group allowed in the archive associated with the archive media class relationship.
VSID_TARGET_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Pointer to the destination archive for media automatically migrated out of this archive media class. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Return Values

`VS_ArchiveMediaClass_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not an archive media class handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_archivemediaclass
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * an Archive Media Class handle.
8  *
9  * PARAMETERS:
10 * h : the Archive Media Class handle to
       print
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    void
         vst_print_archivemediaclass
         (VST_ARCHIVEMEDIACLASS_HANDLE h)
15 #else
16    void vst_print_archivemediaclass(h)
17 VST_ARCHIVEMEDIACLASS_HANDLE h;
18 #endif
19 {
20    VST_ARCHIVE_NAME           Archive;
21    VST_MEDIA_CLASS_NAME
         MediaClass;
22    VST_MEDIA_TYPE_NAME
         MediaType;
23    VST_CAPACITY               Capacity;
24    VST_PERCENT
         MediaClassPercent;
25    VST_ARCHIVE_ACTION_OPTION
         ActionMode;
26    VST_HIGH_MARK              HighMark;
27    VST_LOW_MARK               LowMark;
28    VST_FILL_LEVEL
         FillLevel;
29    VST_PRIORITY
         MigrationPriority;
```

```
30    VST_ARCHIVE_NAME
         TargetArchive;
31    VST_TABLE_HANDLE
         RestrictedComps;
32    int                        i;
33    int                        n;
34    VST_COMPONENT_HANDLE
         Component;
35
36    VS_ArchiveMediaClass_GetFields(h,
37       VSID_ARCHIVE_NAME,
         Archive,
38       VSID_MEDIA_CLASS_NAME,
         MediaClass,
39       VSID_MEDIA_TYPE_NAME,
         MediaType,
40       VSID_CAPACITY,
         &Capacity,
41       VSID_PERCENT,
         &MediaClassPercent,
42       VSID_ARCHIVE_ACTION,
         &ActionMode,
43       VSID_HIGH_MARK,
         &HighMark,
44       VSID_LOW_MARK,
         &LowMark,
45       VSID_FILL_LEVEL,
         &FillLevel,
46       VSID_MIGRATION_PRIORITY,
         &MigrationPriority,
47       VSID_TARGET_ARCHIVE_NAME,
         TargetArchive,
48
         VSID_COMPONENT_HANDLE_TABLE,&Rest
         rictedComps,
49       VSID_ENDFIELD);
50    printf("*** Archive Media Class
         Handle ***\n");
51    printf("Archive Name = %s\n",
         Archive);
52    printf("Media Class Name = %s\n",
         MediaClass);
```

```
53    printf("Media Type Name = %s\n",
         MediaType);
54    printf("Capacity = %d\n", Capacity);
55    printf("Media Class percent = %d\n",
         MediaClassPercent);
56    printf("Archive Action Mode = %d\n",
         ActionMode);
57    printf("High Mark = %d\n", HighMark);
58    printf("Low Mark = %d\n", LowMark);
59    printf("Fill Level = %d\n",
         FillLevel);
60    printf("Migration Priority = %d\n",
         MigrationPriority);
61    printf("Target Archive = %s\n",
         TargetArchive);
62    if (RestrictedComps !=
         (VST_TABLE_HANDLE) NULL)
63    {
64       /* get number of entries */
65
         VS_Table_GetFields(RestrictedComp
         s,
66          VSID_NUMBER_ENTRIES, &n,
67          VSID_ENDFIELD);
68       for (i = 0; i < n; i++)
69       {
70
         VS_Table_GetFields(RestrictedComp
         s,
71               VSID_TABLE_ENTRY, i,
         &Component,
72               VSID_ENDFIELD);
73
         vst_print_component(Component);
74       }
75    }
76 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

The migration policy options for are no action, operator notification, and automatic migration.

When the number of media in an archive media class reaches the high mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy option is set to notify.
- Initiates automatic migration of media if the migration policy is set to migrate.

When the number of media in an archive media class drops to the low mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy is set to notify.
- Terminates automatic migration of media if the migration policy is set to migrate.

See Also

- vsapi(l),
- VS_Archive_GetFields(l),
- VS_ArchiveMediaClass_Create(l),
- VS_ArchiveMediaClass_Destroy(l),
- VS_ArchiveMediaClass_SetFields(l),
- VS_Error_GetFields(l),
- VSCMD_ArchiveQuery(l)

VS_ArchiveMediaClass_SetFields

`VS_ArchiveMediaClass_SetFields` sets the value of one or more fields in an archive media class handle. An archive media class handle is used to pass archive media class information to and from VolServ.

Synopsis

```
VST_BOOLEAN VS_ArchiveMediaClass_SetFields
(VST_ARCHIVEMEDIACLASS
_HANDLE
handle,
"…",
VSID_ENDFIELD )
```

Arguments

- `handle` =Archive media class handle where information is stored.
- `"…"` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_ACTION (VST_ARCHIVE_ACTION_OPTION)	The archive action VolServ is to take when the number of media in the archive media class exceeds the specified high mark threshold. Valid VSID_ARCHIVE_ACTION values are enumerated in the *vs_types.h* file.
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The name of the archive to be associated with the archive media class relationship. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CAPACITY (VST_CAPACITY)	The percentage of the total MediaClass capacity that can be stored in this archive.
VSID_COMPONENT_HANDLE_TABLE (VST_TABLE_HANDLE)	The preferred locations (in table format) for media assigned to this archive media class.
VSID_FILL_LEVEL (VST_FILL_LEVEL)	The number of media currently in this archive media class.
VSID_HIGH_MARK (VST_HIGH_MARK)	The percentage of VSID_CAPACITY above which the specified migration policy option is performed or initiated. This field is applicable only if VSID_ARCHIVE_ACTION is set to VSE_ARCHIVE_ACTION_NOTIFY or VSE_ARCHIVE_ACTION_MIG.

Parameter Type	Description
VSID_LOW_MARK (VST_LOW_MARK)	The percentage of the archive media class capacity below which automatic migration of media out of the archive media class stops. A message is generated to the operator whenever the number of media in the archive media class drops below this threshold. This field is applicable only if VSID_ARCHIVE_ACTION is set to VSE_ARCHIVE_ACTION_NOTIFY or VSE_ARCHIVE_ACTION_MIG.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The name of the MediaClass group to be associated with the archive media class relationship. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	The media type associated with the archive media class. Valid media type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Parameter Type	Description
VSID_MIGRATION_PRIORITY(VST_PRIORITY)	The migration priority to be applied to this archive media class. Increasing the archive media class migration priority results in media being selected from this archive media class archive media class before media from archive media classes with lower migration priorities. Likewise, decreasing the archive media class migration priority results in media being selected from this archive media class after media from archive media classes with higher migration priorities.
VSID_PERCENT (VST_PERCENT)	The percentage of the media assigned to the related MediaClass group allowed in the archive associated with the archive media class relationship.
VSID_TARGET_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The destination archive for media automatically migrated out of this archive media class. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Return Values

VS_ArchiveMediaClass_SetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.

- VSE_ERR_BADHANDLE - Specified handle was not an archive media class handle.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_archivemediaclass_handle
4  *
5  * PURPOSE:
6  * This function tests an
       archivemediaclass handle.
7  *
8  * PARAMETERS:
9  * none
10
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_archivemediaclass_handle(void
         )
14 #else
15    VST_BOOLEAN
         vst_archivemediaclass_handle()
16 #endif
17 {
18    VST_BOOLEAN rc;
19    VST_ARCHIVEMEDIACLASS_HANDLE  h;
```

```
20    VST_ARCHIVE_NAME
         Archive;
21    VST_MEDIA_CLASS_NAME
         MediaClass;
22    VST_MEDIA_TYPE_NAME
         MediaType;
23    VST_CAPACITY
         Capacity;
24    VST_PERCENT
         MediaClassPercent;
25    VST_ARCHIVE_ACTION_OPTION
         ActionMode;
26    VST_HIGH_MARK
         HighMark;
27    VST_LOW_MARK
         LowMark;
28    VST_FILL_LEVEL
         FillLevel;
29    VST_PRIORITY
         MigrationPriority;
30    VST_ARCHIVE_NAME
         TargetArchive;
31
32    /* create the handle */
33    h = VS_ArchiveMediaClass_Create();
34    if (h !=
         (VST_ARCHIVEMEDIACLASS_HANDLE)
         NULL)
35    {
36       /* get the values from the user */
37       printf("** Archive Media Class
         handle **\n");
38       printf("Enter Archive Name ==> ");
39       gets(Archive);
40       printf("Enter Media Class Name ==>
         ");
41       gets(MediaClass);
42       printf("Enter Media Type Name ==>
         ");
43       gets(MediaType);
44       printf("Enter Capacity ==> ");
45       Capacity = atoi(gets(input));
```

```
46        printf("Enter MediaClass Percent
          ==> ");
47        MediaClassPercent =
          atoi(gets(input));
48        printf("Enter Archive Action Mode
          ==> ");
49        ActionMode = atoi(gets(input));
50        printf("Enter High Mark Mode ==>
          ");
51        HighMark = atoi(gets(input));
52        printf("Enter Fill Level Mode ==>
          ");
53        FillLevel = atoi(gets(input));
54        printf("Enter Migration Priority
          ==> ");
55        MigrationPriority =
          atoi(gets(input));
56        printf("Enter Target Archive ==>
          ");
57        gets(TargetArchive);
58        rc =
          VS_ArchiveMediaClass_SetFields(h,
59           VSID_ARCHIVE_NAME,
          Archive,
60           VSID_MEDIA_CLASS_NAME,
          MediaClass,
61           VSID_MEDIA_TYPE_NAME,
          MediaType,
62           VSID_CAPACITY,
          Capacity,
63           VSID_PERCENT,
          MediaClassPercent,
64           VSID_ARCHIVE_ACTION,
          ActionMode,
65           VSID_HIGH_MARK,
          HighMark,
66           VSID_LOW_MARK,
          LowMark,
67           VSID_FILL_LEVEL,
          FillLevel,
68           VSID_MIGRATION_PRIORITY,
          MigrationPriority,
```

```
69          VSID_TARGET_ARCHIVE_NAME,
          TargetArchive,
70          VSID_ENDFIELD);
71        if (rc)
72        {
73
          vst_print_archivemediaclass(h);
74        }
75        VS_ArchiveMediaClass_Destroy(h);
76     }
77     return(rc);
78 }
```

Notes

The migration policy options for are no action, operator notification, and automatic migration.

When the number of media in an archive media class reaches the high mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy option is set to notify.
- Initiates automatic migration of media if the migration policy is set to migrate.

When the number of media in an archive media class drops to the low mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy is set to notify.

- Terminates automatic migration of media if the migration policy is set to migrate.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_ArchiveMediaClass_Create(l),
- VS_ArchiveMediaClass_Destroy(l),
- VS_ArchiveMediaClass_GetFields(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VS_Table_Create(l),
- VS_Table_Destroy(l),
- VS_Table_GetFields(l),
- VS_Table_SetFields(l)

VS_ Command_ Create

VS_Command_Create allocates a VolServ API command handle. A command handle is used to pass command information to and from VolServ.

Synopsis

VST_COMMAND_HANDLE VS_Command_Create (void)

Arguments

None

Return Values

VS_Command_Create returns:

- A command handle, if one can be allocated.
- NULL, if a command handle cannot be allocated. An appropriate error code is set in VSG_Error.
- VSE_ERR_OUTOFMEM - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_command_handle
4  *
5  * PURPOSE:
6  * This function tests a command handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_command_handle(void)
14 #else
15    VST_BOOLEAN vst_command_handle(void)
16 #endif
```

```
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_COMMAND_HANDLE   h;
20    VST_REQUEST_ID       requestid;
21    VST_REQUEST_TYPE     reqtype;
22    VST_RETRY_LIMIT      retrylimit;
23    VST_TIME_OUT         timeout;
24    VST_STATUS_WAIT_FLAG waitflag;
25
26    /* create the handle */
27    h = VS_Command_Create();
28    if (h != (VST_COMMAND_HANDLE) NULL)
29    {
30       /* get values from user */
31       printf("*** Command Handle
         ***\n");
32       printf("Enter Request ID ==> ");
33       requestid = atol(gets(input));
34       printf("Enter Request type ==> ");
35       reqtype = atol(gets(input));
36       printf("Enter Retry Limit ==> ");
37       retrylimit = atol(gets(input));
38       printf("Enter Timeout Value ==>
         ");
39       timeout = atol(gets(input));
40
41       /* set fields in handle */
42       rc = VS_Command_SetFields(h,
43                VSID_REQUEST_ID,
         requestid,
44                VSID_REQUEST_TYPE,
         reqtype,
45                VSID_RETRY_LIMIT,
         retrylimit,
46                VSID_TIMEOUT_VALUE,
         timeout,
47                VSID_STATUS_WAIT_FLAG,
         waitflag,
48                VSID_ENDFIELD);
49       if (rc)
50       {
```

```
51          /* print the handle and destroy
        it */
52          vst_print_command(h);
53       }
54       VS_Command_Destroy(h);
55    }
56    return(rc);
57 }
```

Notes

A command handle must be used for only one outstanding command at any given time.

See Also

- vsapi(l),
- VS_Command_Destroy(l),
- VS_Command_GetFields(l),
- VS_Command_SetFields(l),
- VS_Error_GetFields(l)

VS_ Command_ Destroy

VS_Command_Destroy deallocates a command handle that was allocated with VS_Command_Create. A command handle is used to pass command information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Command_Destroy
(VST_COMMAND_HANDLE
cmdhandle)

Arguments

- cmdhandle = Command handle to be destroyed.

Return Values

VS_Command_Destroy returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a command handle.
- VSE_ERR_EXECUTING - Final status has not been returned for the command associated with the specified command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_command_handle
4  *
5  * PURPOSE:
6  * This function tests a command handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_command_handle(void)
14 #else
15    VST_BOOLEAN vst_command_handle(void)
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_COMMAND_HANDLE   h;
20    VST_REQUEST_ID       requestid;
21    VST_REQUEST_TYPE     reqtype;
22    VST_RETRY_LIMIT      retrylimit;
23    VST_TIME_OUT         timeout;
24    VST_STATUS_WAIT_FLAG waitflag;
25
26    /* create the handle */
27    h = VS_Command_Create();
28    if (h != (VST_COMMAND_HANDLE) NULL)
29    {
30       /* get values from user */
31       printf("*** Command Handle
         ***\n");
32       printf("Enter Request ID ==> ");
33       requestid = atol(gets(input));
34       printf("Enter Request type ==> ");
35       reqtype = atol(gets(input));
36       printf("Enter Retry Limit ==> ");
37       retrylimit = atol(gets(input));
38       printf("Enter Timeout Value ==>
         ");
```

```
39       timeout = atol(gets(input));
40
41       /* set fields in handle */
42       rc = VS_Command_SetFields(h,
43                VSID_REQUEST_ID,
         requestid,
44                VSID_REQUEST_TYPE,
         reqtype,
45                VSID_RETRY_LIMIT,
         retrylimit,
46                VSID_TIMEOUT_VALUE,
         timeout,
47                VSID_STATUS_WAIT_FLAG,
         waitflag,
48                VSID_ENDFIELD);
49       if (rc)
50       {
51          /* print the handle and destroy
         it */
52          vst_print_command(h);
53       }
54       VS_Command_Destroy(h);
55    }
56    return(rc);
57 }
```


Notes

A command handle should be used for only one outstanding command at any given time.

After `VS_Command_Destroy` has been called for a command handle, that handle is no longer valid and should not be used.

If final status has not been received for a command, the API software fails a `VS_Command_Destroy` request for the associated command handle.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_GetFields(l),
- VS_Command_SetFields(l),
- VS_Error_GetFields(l)

VS_ Command_ GetError-Fields

`VS_Command_GetErrorFields` retrieves information associated with the command's error handle. It can be used in place of the `VS_Error_GetFields` routine when the user does not want to retrieve the error handle explicitly from the command handle.

Synopsis

VST_BOOLEAN VS_Command_GetErrorFields
(VST_COMMAND_HANDLE cmdhandle,

Arguments

- cmdhandle = Command handle where the status information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by Pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following Parameters section.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ERROR_CODE (VST_ERROR_CODE)	Pointer to the error code for the given error.
VSID_ERROR_FILE (VST_ERROR_FILE)	The name of the source file where the error occurred (API internal errors only).
VSID_ERROR_LINE (int *)	Pointer to the source line number where the error occurred (API internal errors only).

Parameter Type	Description
VSID_ERROR_NUMBER (VST_ERROR_NUMCODE *)	Pointer to the field that indicates which error occurred.
VSID_ERROR_OBJECT (VST_ERROR_OBJCODE *)	Pointer to the field that indicates the location of the error.

Return Values

`VS_Command_GetErrorFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Notes

`VS_Error_GetErrorFields` returns

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - The appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not an error handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

See Also

- VS_Command_Create,
- VS_Command_Destroy,
- VS_Command_GetFields,
- VS_Command_SetFields,
- VS_Error_GetFields

VS_ Command_ GetFields

VS_Command_GetFields retrieves information associated with a command handle. A command handle is used to pass command information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Command_GetFields
(VST_COMMAND_HANDLE cmdhandle,
"…",
VSID_ENDFIELD)

Arguments

- cmdhandle = Command handle where information is retrieved.
- "..." = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ERROR_HANDLE (VST_ERROR_HANDLE *)	Pointer to the error handle associated with this command.
VSID_REQUEST_ID (VST_REQUEST_ID *)	Pointer to the request identifier associated with this command.

Parameter Type	Description
VSID_REQUEST_TYPE (VST_REQUEST_TYPE *)	Pointer to the type of command. Valid VSID_REQUEST_TYPE values are enumerated in the *vs_type.h* file.
VSID_STATUS_HANDLE (VST_STATUS_HANDLE *)	Pointer to the status handle associated with this command.

Return Values

VS_Command_GetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_print_command
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a command handle.
8  *
9  * PARAMETERS:
10 * h : the command handle to print
11 *
```

```
12 ************************************
       *********/
13 #ifdef ANSI_C
14    void
         vst_print_command(VST_COMMAND_HAN
         DLE h)
15 #else
16    void vst_print_command(h)
17    VST_COMMAND_HANDLE h;
18 #endif
19 {
20    VST_REQUEST_ID       requestid;
21    VST_REQUEST_TYPE     reqtype;
22    VST_RETRY_LIMIT      retrylimit;
23    VST_TIME_OUT         timeout;
24    VST_STATUS_WAIT_FLAG waitflag;
25    VST_ERROR_HANDLE     eh;
26    VST_STATUS_HANDLE    sh;
27
28    VS_Command_GetFields(h,
29                VSID_REQUEST_ID,
         &requestid,
30                VSID_REQUEST_TYPE,
         &reqtype,
31                VSID_RETRY_LIMIT,
         &retrylimit,
32                VSID_TIMEOUT_VALUE,
         &timeout,
33                VSID_STATUS_WAIT_FLAG,
         &waitflag,
34                VSID_ERROR_HANDLE,
         &eh,
35                VSID_STATUS_HANDLE,
         &sh,
36                VSID_ENDFIELD);
37
38    printf(" ****Command Handle
         Information****\n\n");
39    printf("RequestID = %luRequestType =
         %d\n",requestid,reqtype);
40    printf("Retry Limit = %dTime Out =
         %d\n",retrylimit,timeout);
```

```
41    printf("StatusWait =
         %d\n",waitflag);
42
43    vst_print_status(sh);
44    vst_print_error(eh);
45 }
```

Notes

The command's status is kept in its status handle.

The command's error is kept in its error handle.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Command_SetFields(l),
- VS_Error_GetFields(l)

VS_Command_GetStatus-Fields

VS_Command_GetStatusFields retrieves information associated with the command's status handle. It can be used in place of the VS_Status_GetFields routine when the user does not want to retrieve the status handle explicitly from the command handle

Synopsis

`VST_BOOLEAN VS_Command_GetStatusFields` (`VST_COMMAND_HANDLE` cmdhandle, "…", `VSID_ENDFIELD`)

Arguments

- `cmdhandle` = Command handle from where the status information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by Pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following Parameters section.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ACTION_CODE (VST_ACTION_CODE *)	Pointer to the first entry in the action code table.
VSID_ACTION_CODE_ENTRY (int)	Index of the appropriate entry in the action code table.
(VST_ACTION_CODE *)	Pointer to the appropriate entry in the action code table.
VSID_ACTION_CODE_TABLE (VST_TABLE_HANDLE *)	Pointer to the action code table associated with this status.
VSID_ARCHIVE_HANDLE (VST_ARCHIVE_HANDLE *)	Pointer to the first archive handle in the archive handle table.
VSID_ARCHIVE_HANDLE_ENTRY (int)	Index of the appropriate archive handle in the archive handle table.
(VST_ARCHIVE_HANDLE *)	Pointer to the appropriate archive handle in the archive handle table.
VSID_ARCHIVE_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the archive handle table associated with this status.
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Pointer to the name of the archive associated with this status. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ARCHIVEMEDIACLASS_HANDLE (VST_ARCHIVE_MEDIACLAS_HANDLE *)	Pointer to the first archive media class handle in the archive media class table.
VSID_ARCHIVEMEDIACLASS_HANDLE_ ENTRY (int)	Index of the appropriate archive media class handle in the archive media class handle table.
(VST_ARCHIVEMEDIACLASS_HANDLE *)	Pointer to the appropriate archive media class handle in the archive media class handle table.

Parameter Type	Description
VSID_ARCHIVEMEDIACLASS_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the archive media class handle table associated with this status.
VSID_COMPONENT_HANDLE (VST_COMPONENT_HANDLE *)	Pointer to the first component handle in the component handle table.
VSID_COMPONENT_HANDLE_ENTRY (int) (VST_COMPONENT_HANDLE *)	Index of the appropriate component handle in the component handle table. Pointer to the appropriate component handle in the component handle table.
VSID_COMPONENT_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the component handle table associated with this status.
VSID_COMP_ID (VST_COMPONENT_ID *)	Pointer to the first component identifier in the component identifier table.
VSID_COMP_ID_ENTRY (int) (VST_COMPONENT_ID *)	Index of the appropriate component identifier in the component identifier table. Pointer to the appropriate component identifier in the component identifier table.
VSID_COMP_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the component identifier table associated with this status.
VSID_COMP_STATE (VST_COMPONENT_STATE *)	Pointer to the component state for this status. Valid `VSID_COMP_STATE` values are enumerated in the `vs_types.h` file.
VSID_CONNECT_HANDLE (VST_CONNECT_HANDLE *)	Pointer to the first entry in the connect handle table.
VSID_CONNECT_HANDLE_ENTRY (int)	Index of the appropriate connect handle in the connect handle table.
(VST_CONNECT_HANDLE *)	Pointer to the appropriate connect handle in the connect handle table.
VSID_CONNECT_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the connect handle table associated with this status.

Parameter Type	Description
VSID_DRIVE_HANDLE (VST_DRIVE_HANDLE *)	Pointer to the first drive handle in the drive handle table.
VSID_DRIVE_HANDLE_ENTRY (int)	Index the appropriate drive handle in the drive handle table.
(VST_DRIVE_HANDLE *)	Pointer to the appropriate drive handle in the drive handle table.
VSID_DRIVE_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the drive handle table associated with this status.
VSID_DRIVE_ID (VST_DRIVE_ID *)	Pointer to the first drive identifier in the drive identifier table.
VSID_DRIVE_ID_ENTRY (int)	Index of the appropriate drive identifier in the drive identifier table.
(VST_DRIVE_ID *)	Pointer to the appropriate drive identifier in the drive identifier table.
VSID_DRIVE_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the drive identifier table associated with this status.
VSID_DRIVEPOOL_HANDLE (VST_DRIVE_POOL_HANDLE *)	Pointer to the first drive pool handle in the drive pool handle table.
VSID_DRIVEPOOL_HANDLE_ENTRY (int)	Index of the appropriate drive pool handle in the drive pool handle table.
(VST_DRIVEPOOL_HANDLE *)	Pointer to the appropriate drive pool handle in the drive pool handle table.
VSID_DRIVEPOOL_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the drive pool handle table associated with this status.
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	Pointer to the name of the drive pool group.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID *)	Pointer to the identifier of the enterprise, if any, to receive command status.

Parameter Type	Description
VSID_ERROR_CODE (VST_VOLERR_CODE *)	Pointer to the first error code in the error code table.
VSID_ERROR_CODE_ENTRY (int)	Index of the appropriate error code in the error code table.
(VST_VOLERR_CODE *)	Pointer to the appropriate error code in the error code table.
VSID_ERROR_TABLE (VST_TABLE_HANDLE *)	Pointer to the error code table associated with this status.
VSID_FIELD (int *)	Pointer to the first field in the user-defined media statistics field table.
VSID_FIELD_ENTRY (int)	Index of the appropriate field in the user-defined media statistics field table.
(int *)	Pointer to the appropriate field in the user-defined media statistics field table.
VSID_FIELD_TABLE (VST_TABLE_HANDLE *)	Pointer to the user-defined media statistics field table associated with this status.
VSID_LOCK_ID (VST_LOCK_ID *)	Pointer to the lock identifier associated with this status.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Pointer to the MediaClass name associated with this status.
VSID_MEDIA_HANDLE (VST_MEDIA_HANDLE *)	Pointer to the first media handle in the media handle table.
VSID_MEDIA_HANDLE_ENTRY (int)	Index of the appropriate media handle in the media handle table.
(VST_MEDIA_HANDLE *)	Pointer to the appropriate media handle in the media handle table.
VSID_MEDIA_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the media handle table associated with this status.
VSID_MEDIA_ID (VST_MEDIA_ID)	Pointer to the first media identifier in the media identifier table.

Parameter Type	Description
VSID_MEDIA_ID_ENTRY (int)	Index of the appropriate entry in the media identifier table.
(VST_MEDIA_ID *)	Pointer to the appropriate media identifier in the media identifier table.
VSID_MEDIA_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the media identifier table associated with this status.
VSID_MEDIACLASS_HANDLE (VST_MEDIACLASS_HANDLE *)	Pointer to the first MediaClass handle in the MediaClass handle table.
VSID_MEDIACLASS_HANDLE_ENTRY (int)	Index of the appropriate MediaClass handle in the MediaClass handle table
(VST_MEDIACLASS_HANDLE *)	Pointer to the appropriate MediaClass handle in the MediaClass handle table.
VSID_MEDIACLASS_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the MediaClass handle table associated with this status.
VSID_MEDIATYPE_HANDLE (VST_MEDIATYPE_HANDLE *)	Pointer to the first media type handle in the media type handle table.
VSID_MEDIATYPE_HANDLE_ENTRY (int)	Index of the appropriate media type handle in the media type handle table.
(VST_MEDIATYPE_HANDLE *)	Pointer to the appropriate media type handle in the media type handle table.
VSID_MEDIATYPE_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the media type handle table associated with this status.
VSID_NUMBER_ACTION_CODES (int *)	Pointer to the number of action codes in the action code table.
VSID_NUMBER_ARCHIVE_HANDLES (int *)	Pointer to the number of archive handles in the archive handle table.
VSID_NUMBER_COMP_IDS (int *)	Pointer to the number of component ids in the component identifier table.

Parameter Type	Description
VSID_NUMBER_COMPONENT_HANDLES (int *)	Pointer to the number of component handles in the component handle table.
VSID_NUMBER_CONNECT_HANDLES (int *)	Pointer to the number of connect handles in the connect handle table.
VSID_NUMBER_DRIVE_HANDLES (int *)	Pointer to the number of drive handles in the drive handle table.
VSID_NUMBER_DRIVE_IDS (int *)	Pointer to the number of drive ids in the drive id table.
VSID_NUMBER_DRIVEPOOL_HANDLES (int *)	Pointer to the number of drive pool handles in the drive pool handle table.
VSID_NUMBER_ERROR_CODES (int *)	Pointer to the number of error codes in the error code table.
VSID_NUMBER_FIELDS (int *)	Pointer to the number of field ids in the field identifier table.
VSID_NUMBER_MEDIA_HANDLES (int *)	Pointer to the number of media handles present in the media handle table.
VSID_NUMBER_MEDIA_IDS (int *)	Pointer to the number of media ids in the media id table.
VSID_NUMBER_MEDIACLASS_HANDLES (int *)	Pointer to the number of media class handles in the media class handle table.
VSID_NUMBER_MEDIATYPE_HANDLES (int *)	Pointer to the number of media type handles in the media type handle table.
VSID_NUMBER_REQUEST_IDS (int *)	Pointer to the number of request ids present in the request id table.
VSID_NUMBER_REQUEST_HANDLES (int *)	Pointer to the number of request handles in the request handle table.
VSID_PID (VST_PID *)	Pointer to the VolServ identifier for the Ping status.
VSID_QRY_ENTERPRISE_ID (VST_ENTERPRISE_ID *)	Pointer to the enterprise identifier, if any, for the Connect Query command.

Parameter Type	Description
VSID_QRY_OPTION (VST_QRY_OPTION *)	Pointer to the query option for this status.
VSID_REQUEST_HANDLE (VST_REQUEST_HANDLE *)	Pointer to the first request handle in the request handle table.
VSID_REQUEST_HANDLE_ENTRY (int)	Index of the appropriate request handle in the request handle table.
(VST_REQUEST_HANDLE *)	Pointer to the appropriate request handle in the request handle table.
VSID_REQUEST_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the request handle table associated with this status.
VSID_REQUEST_ID (VST_REQUEST_ID *)	Pointer to the request identifier of the target command for a Cancel or Reprioritize request.
VSID_SEQUENCE_NUM (int *)	Pointer to the sequence number of this status. Initial status for a command request is sequence number 0. Sequence numbers for subsequent statuses for the same command request are assigned as one-up numbers. For example, the first intermediate status (if there is an intermediate status) or the final status (if there is no intermediate status) is sequence number 1.
VSID_SEQUENCE_TABLE (VST_TABLE_HANDLE *)	Pointer to the sequence numbers (in table format) of the statuses received for this command.
VSID_STATUS_CODE (VST_STATUS_CODE *)	Pointer to the status code for this status. Indicates whether the command was successful or failed. Valid `VSID_STATUS_CODE` values are enumerated in the *vs_types.h* file.
VSID_STATUS_TYPE (VST_STATUS_TYPE *)	Pointer to the status type (intermediate or final) for this status. Valid `VSID_STATUS_TYPE` values are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_TARGET_ENTERPRISE_ID (VST_ENTERPRISE_ID *)	Pointer to the enterprise identifier for a ConnectQuery or Disconnect command.
VSID_USER_FIELD (VST_USER_FIELD)	Pointer to the user field contents for the associated command. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for each command. Neither the API software nor VolServ uses USER_FIELD.
VSID_WAIT_REASON (VST_WAIT_REASON *)	Pointer to the wait reason for an intermediate status. Valid VSID_WAIT_REASON values are enumerated in the *vs_types.h* file.

Return Values

VS_Command_GetStatusFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Notes

The command's status is kept in its status handle, and the command's error is kept in its error handle.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- VS_Command_Create,
- VS_Command_Destroy,
- VS_Command_GetFields,
- VS_Command_SetFields,
- VS_Status_GetFields

VS_ Command_ SetFields

`VS_Command_Setfields` sets the value of one or more fields in a command handle. A command handle is used to pass command information to and from VolServ.

Synopsis

```
VST_BOOLEAN VS_Command_Setfields
( VST_COMMAND_HANDLE cmdhandle,
"…",
VSID_ENDFIELD )
```

Arguments

- `cmdhandle` = Command handle where information is stored.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` =Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ERROR_HANDLE (VST_ERROR_HANDLE)	Error handle associated with this command.
VSID_REQUEST_ID (VST_REQUEST_ID)	Request identifier associated with this command.
VSID_REQUEST_TYPE (VST_REQUEST_TYPE)	Type of command. Valid `VSID_REQUEST_TYPE` values are enumerated in the *vs_type.h* file.
VSID_STATUS_HANDLE (VST_STATUS_HANDLE)	Status handle associated with this command.

Return Values

`VS_Command_Setfields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADHANDLE` - Specified handle was not a command handle.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_command_handle
4  *
5  * PURPOSE:
6  * This function tests a command handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_command_handle(void)
14 #else
15    VST_BOOLEAN vst_command_handle(void)
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_COMMAND_HANDLE   h;
20    VST_REQUEST_ID       requestid;
21    VST_REQUEST_TYPE     reqtype;
22    VST_RETRY_LIMIT      retrylimit;
23    VST_TIME_OUT         timeout;
24    VST_STATUS_WAIT_FLAG waitflag;
25
26    /* create the handle */
27    h = VS_Command_Create();
28    if (h != (VST_COMMAND_HANDLE) NULL)
29    {
30       /* get values from user */
31       printf("*** Command Handle
         ***\n");
32       printf("Enter Request ID ==> ");
33       requestid = atol(gets(input));
34       printf("Enter Request type ==> ");
35       reqtype = atol(gets(input));
36       printf("Enter Retry Limit ==> ");
37       retrylimit = atol(gets(input));
38       printf("Enter Timeout Value ==>
         ");
```

```
39        timeout = atol(gets(input));
40
41        /* set fields in handle */
42        rc = VS_Command_SetFields(h,
43                  VSID_REQUEST_ID,
          requestid,
44                  VSID_REQUEST_TYPE,
          reqtype,
45                  VSID_RETRY_LIMIT,
          retrylimit,
46                  VSID_TIMEOUT_VALUE,
          timeout,
47                  VSID_STATUS_WAIT_FLAG,
          waitflag,
48                  VSID_ENDFIELD);
49        if (rc)
50        {
51           /* print the handle and destroy
          it */
52           vst_print_command(h);
53        }
54        VS_Command_Destroy(h);
55     }
56     return(rc);
57  }
```


Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Command_GetFields(l),

- VS_Error_GetFields(l)

VS_ Component_ Create

VS_Component_Create allocates a VolServ API component handle. A component handle is used to pass component information to and from VolServ.

Synopsis

VST_COMPONENT_HANDLE VS_Component_Create
(void)

Arguments

None

Return Values

VS_Component_Create returns:

- A component handle, if one can be allocated.
- NULL, if a component handle cannot be allocated. An appropriate error code is set in VSG_Error.
- VSE_ERR_OUTOFMEM - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  *FUNCTION:
       vst_modarchivemediaclass_execute
4  *
5  * PURPOSE:
6  * This executes the
       VSCMD_ModifyArchiveMediaClass
7  * API all.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13
```

```
14 #ifdef ANSI_C
15    VST_BOOLEAN
         vst_modarchivemediaclass_execute(
         void)
16 #else
17    VST_BOOLEAN
         vst_modarchivemediaclass_execute(
         )
18 #endif
19 {
20    int                          i;
21    int count;
22    VST_BOOLEAN                  rc =
         VSE_FALSE;
23    VST_ARCHIVE_NAME             archive;
24    VST_MEDIA_CLASS_NAME
         mediaclass;
25    VST_CAPACITY                 capacity;
26    VST_ARCHIVE_ACTION_OPTION    action;
27    VST_HIGH_MARK                highmark;
28    VST_LOW_MARK                 lowmark;
29    VST_PRIORITY                 migpri;
30    VST_ARCHIVE_NAME
         targetarchive;
31    VST_TABLE_HANDLE
         comphandletable;
32    VST_COMPONENT_HANDLE
         comphandle;
33    VST_COMP_TYPE     CompType =
         VSE_COMPTYPE_COLUMN;
34    VST_COMPONENT_ID             CompID;
35    VST_COMMAND_HANDLE           cmd;
36
37    bzero ( CompID, sizeof (
         VST_COMPONENT_ID ) );
38    /* get parameters from user */
39    printf("Modify archive media class
         parameters \n" );
40    printf("The archive media class must
         exist. \n");
41    printf("Enter Archive Name ==> " );
42    gets( archive );
```

```
43    printf("Enter Media Class Name ==> "
         );
44    gets( mediaclass );
45    printf("Enter Capacity Percent ==> "
         );
46    capacity = atoi(gets(input));
47    printf("Enter Archive action option
         (0-none/1-mig/2-notify) ==>" );
48    action = atoi(gets(input));
49    printf("Enter High Mark Percentage
         ==> " );
50    highmark = atoi(gets(input));
51    printf("Enter Low Mark Percentage ==>
         " );
52    lowmark = atoi(gets(input));
53
54    if ( action == VSE_ARCHIVE_ACTION_MIG
         )
55    {
56       printf("Enter Target Archive ==> "
         );
57       gets( targetarchive );
58       printf("Enter Migration Priority
         == > " );
59       migpri = atoi(gets(input));
60       /* These only need to be set when
         */
61       /* migration is used. */
62
         VSCMD_ModifyArchiveMediaClass_Set
         Defaults (
63             VSID_TARGET_ARCHIVE_NAME,
         targetarchive,
64             VSID_MIGRATION_PRIORITY,
         migpri,
65             VSID_ENDFIELD );
66    }
67
68    printf("How many prefered placements
         (0 to skip): ");
69    count = atoi(gets(input));
70    if (count > 0)
```

```
71    {
72       comphandletable =
         VS_Table_Create(VSE_COMPONENT_HAN
         DLE, count);
73       if (comphandletable ==
         (VST_TABLE_HANDLE) NULL)
74       {
75          return (VSE_FALSE);
76       }
77       for (i = 0; i < count; i++)
78       {
79          printf("Enter row #%d:", i +
         1);
80          CompID[0] = (short)
         atoi(gets(input));
81          printf("Enter column #%d:", i +
         1);
82          CompID[1] = (short)
         atoi(gets(input));
83          CompID[2] = 0;
84          CompID[3] = 0;
85          comphandle =
         VS_Component_Create();
86
         VS_Component_SetFields(comphandle
         ,
87                      VSID_COMP_TYPE,
         CompType,
88                      VSID_COMP_ID,
         CompID,
89                      VSID_ENDFIELD);
90
         VS_Table_AddEntry(comphandletable
         ,comphandle);
91       }
92
         VSCMD_ModifyArchiveMediaClass_Set
         Defaults(
93          VSID_COMPONENT_HANDLE_TABLE,
         comphandletable,
94          VSID_ENDFIELD);
95    }
```

```
96    /* create the command handle */
97    /* Note that the command handle is
         not */
98    /* destroyed in this routine, */
99    /* but in vst_dispatch */
100   /* when final status is received. */
101   cmd = VS_Command_Create();
102   if (cmd != (VST_COMMAND_HANDLE )NULL)
103   {
104      /* Send the command to the VolServ
         software. */
105      /* Note that status is not
         processed here. */
106      /* Instead, it is processed in the
         vst_dispatch */
107      /* routine. Also, note that
         default values such */
108      /* as timeout, value retry limit
         and priority */
109      /* are set as default parameters.
         */
110      rc =
         VSCMD_ModifyArchiveMediaClass(cmd
         ,
111               VSID_ARCHIVE_NAME,
         archive,
112               VSID_MEDIA_CLASS_NAME,
         mediaclass,
113               VSID_HIGH_MARK,
         highmark,
114               VSID_LOW_MARK,
         lowmark,
115               VSID_CAPACITY,
         capacity,
116               VSID_ENDFIELD);
117   }
118   return ( rc );
119}
```

Notes None

See Also

- vsapi(l),
- VS_Component_Destroy(l),
- VS_Component_GetFields(l),
- VS_Component_SetFields(l),
- VS_Error_GetFields(l)

VS_ Component_ Destroy

VS_Component_Destroy deallocates a component handle that was allocated with VS_Component_Create. A component handle is used to pass component information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Component_Destroy
(VST_COMPONENT_HANDLE handle)

Arguments

- handle = Component handle to be destroyed.

Return Values

VS_Component_Destroy returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a component handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_component_handle
4  *
5  * PURPOSE:
6  * This function tests the component
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_component_handle(void)
14 #else
15    VST_BOOLEAN vst_component_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_TRUE;
19    VST_COMPONENT_HANDLE    comph;
20    VST_COMP_TYPE           comptype;
21    VST_COMPONENT_ID        id;
22
23    comph = VS_Component_Create();
24    if (comph != (VST_COMPONENT_HANDLE)
         NULL)
25    {
26       printf("enter component type
         ==>");
27       comptype = atoi(gets(input));
28       printf("enter 4 values for the
         component id (e.g. 3 2 4 1) ==>");
29       scanf("%hd %hd %hd %hd", &id[0],
         &id[1], &id[2], &id[3]);
30       gets(input);
31       VS_Component_SetFields(comph,
32                      VSID_COMP_TYPE,
         comptype,
33                      VSID_COMP_ID,
         id,
34                      VSID_ENDFIELD);
35       vst_print_component(comph);
36       VS_Component_Destroy(comph);
37    }
38    else
39    {
40       rc = VSE_FALSE;
41    }
42    return(rc);
43 }
```

Notes

After `VS_Component_Destroy` has been called for a component handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Component_Create(l),
- VS_Component_GetFields(l),
- VS_Component_SetFields(l),
- VS_Error_GetFields(l)

VS_ Component_ GetFields

VS_Component_GetFields retrieves information from a component handle. The component handle is used to pass component information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Component_GetFields
(VST_COMPONENT_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- handle = Component handle for which information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_COMP_ID (VST_COMP_ID *)	Pointer to the identifier of the component.
VSID_COMP_TYPE (VST_COMP_TYPE *)	Pointer to the type of this component.

Return Values

`VS_Component_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a component handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_component
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a component handle.
8  *
9  * PARAMETERS:
10 * h : the component handle to print
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    void
         vst_print_component(VST_COMPONENT
         _HANDLE h)
15 #else
16    void vst_print_component(h)
17    VST_COMPONENT_HANDLE h;
18 #endif
19 {
20    VST_COMP_TYPE     CompType;
```

```
21    VST_COMPONENT_ID  CompID;
22    int               i;
23
24    VS_Component_GetFields(h,
25                   VSID_COMP_TYPE,
         &CompType,
26                   VSID_COMP_ID,
         CompID,
27                   VSID_ENDFIELD);
28    printf("******* Component Handle
         *******\n");
29    printf("Component Type = %d\n",
         CompType);
30    printf("Component ID = ");
31    for (i = 0; i < VSD_MAX_COMPONENT_ID;
         i++)
32    {
33       printf("%d", CompID[i]);
34       if (i < VSD_MAX_COMPONENT_ID - 1)
35       {
36          printf(", ");
37       }
38    }
39    printf("\n");
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Component_Create(l),
- VS_Component_Destroy(l),
- VS_Component_SetFields(l),

- VS_Error_GetFields(l),
- VS_Table_Create(l),
- VS_Table_SetFields(l),
- VS_TableAddEntry(l),
- VSCMD_CreateArchiveMediaClass(l),
- VSCMD_ModifyArchiveMediaClass(l)

VS_ Component_ SetFields

`VS_Component_SetFields` sets the value of one or more fields in a VolServ API component handle. A component handle is used to pass component information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Component_SetFields
(VST_COMPONENT_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Component handle where information is stored.
- `"…"` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_COMP_ID (VST_COMP_ID)	Identifier of the component.
VSID_COMP_TYPE (VST_COMP_TYPE)	Type of this component. Valid `VSID_COMP_TYPE` values are enumerated in the *vs_types.h* file.

Return Values

VS_Component_SetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a component handle.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  *FUNCTION:
       vst_modarchivemediaclass_execute
4  *
5  * PURPOSE:
6  * This executes the
       VSCMD_ModifyArchiveMediaClass
7  * API all.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
       vst_modarchivemediaclass_execute(
       void)
15 #else
```

```
16    VST_BOOLEAN
         vst_modarchivemediaclass_execute(
         )
17 #endif
18 {
19    int                         i;
20    int count;
21    VST_BOOLEAN                 rc =
         VSE_FALSE;
22    VST_ARCHIVE_NAME            archive;
23    VST_MEDIA_CLASS_NAME
         mediaclass;
24    VST_CAPACITY                capacity;
25    VST_ARCHIVE_ACTION_OPTION   action;
26    VST_HIGH_MARK               highmark;
27    VST_LOW_MARK                lowmark;
28    VST_PRIORITY                migpri;
29    VST_ARCHIVE_NAME
         targetarchive;
30    VST_TABLE_HANDLE
         comphandletable;
31    VST_COMPONENT_HANDLE
         comphandle;
32    VST_COMP_TYPE     CompType =
         VSE_COMPTYPE_COLUMN;
33    VST_COMPONENT_ID            CompID;
34    VST_COMMAND_HANDLE          cmd;
35
36    bzero ( CompID, sizeof (
         VST_COMPONENT_ID ) );
37    /* get parameters from user */
38    printf("*** Modify Archive Media
         Class parameters ***\n" );
39    printf("*** The archive media class
         must exist. ***\n");
40    printf("Enter Archive Name ==> " );
41    gets( archive ); archive media class
42    printf("Enter Media Class Name ==> "
         );
43    gets( mediaclass );
44    printf("Enter Capacity Percent ==> "
         );
```

```
45    capacity = atoi(gets(input));
46    printf("Enter Archive action option
         (0-none/1-mig/2-notify) ==>" );
47    action = atoi(gets(input));
48    printf("Enter High Mark Percentage
         ==> " );
49    highmark = atoi(gets(input));
50    printf("Enter Low Mark Percentage ==>
         " );
51    lowmark = atoi(gets(input));
52
53    if ( action == VSE_ARCHIVE_ACTION_MIG
         )
54    {
55       printf("Enter Target Archive ==> "
         );
56       gets( targetarchive );
57       printf("Enter Migration Priority
         == > " );
58       migpri = atoi(gets(input));
59       /* These only need to be set when
         migration */
60       /* is used. */
61
         VSCMD_ModifyArchiveMediaClass_Set
         Defaults (
62          VSID_TARGET_ARCHIVE_NAME,
         targetarchive,
63          VSID_MIGRATION_PRIORITY,
         migpri,
64          VSID_ENDFIELD );
65    }
66
67    printf("How many prefered placements
         (0 to skip): ");
68    count = atoi(gets(input));
69    if (count > 0)
70    {
71       comphandletable =
         VS_Table_Create(VSE_COMPONENT_HAN
         DLE, count);
```

```
72       if (comphandletable ==
         (VST_TABLE_HANDLE) NULL)
73       {
74          return (VSE_FALSE);
75       }
76       for (i = 0; i < count; i++)
77       {
78          printf("Enter row #%d:", i +
         1);
79          CompID[0] = (short)
         atoi(gets(input));
80          printf("Enter column #%d:", i +
         1);
81          CompID[1] = (short)
         atoi(gets(input));
82          CompID[2] = 0;
83          CompID[3] = 0;
84          comphandle =
         VS_Component_Create();
85
         VS_Component_SetFields(comphandle
         ,
86                  VSID_COMP_TYPE,
         CompType,
87                  VSID_COMP_ID,
         CompID,
88                  VSID_ENDFIELD);
89
         VS_Table_AddEntry(comphandletable
         ,comphandle);
90       }
91
         VSCMD_ModifyArchiveMediaClass_Set
         Defaults(
92          VSID_COMPONENT_HANDLE_TABLE,
         comphandletable,
93          VSID_ENDFIELD);
94    }
95    /* create the command handle */
96    /* Note that the command handle is
         not */
```

```
97    /* destroyed in this routine, but in
         */
98    /* vst_dispatch when final status is
         received. */
99    cmd = VS_Command_Create();
100   if (cmd != (VST_COMMAND_HANDLE )NULL)
101   {
102      /* Send the command to the VolServ
         software. */
103      /* Note that status is not
         processed here. */
104      /* Instead, it is processed in the
         */
105      /* vst_dispatch routine. Also,
         note that */
106      /* default values such as timeout,
         value */
107      /* retry limit and priority are
         set as */
108      /* default parameters. */
109      rc =
         VSCMD_ModifyArchiveMediaClass(cmd
         ,
110               VSID_ARCHIVE_NAME,
         archive,
111               VSID_MEDIA_CLASS_NAME,
         mediaclass,
112               VSID_HIGH_MARK,
         highmark,
113               VSID_LOW_MARK,
         lowmark,
114               VSID_CAPACITY,
         capacity,
115               VSID_ENDFIELD);
116   }
117   return ( rc );
118}
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Component_Create(l),
- VS_Component_Destroy(l),
- VS_Component_GetFields(l),
- VS_Error_GetFields(l),
- VS_Table_Create(l),
- VS_Table_SetFields(l),
- VS_TableAddEntry(l)

VS_Connect_Create

VS_Connect_Create allocates a VolServ API connect handle. A connect handle is used to pass connect information to and from VolServ.

Synopsis

VST_CONNECT_HANDLE VS_Connect_Create (void)

Arguments

None

Return Values

VS_Connect_Create returns:

- A connect handle, if one can be allocated.
- NULL, if a connect handle cannot be allocated. An appropriate error code is set in VSG_Error.
- VSE_ERR_OUTOFMEM - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_connect_handle
4  *
5  * PURPOSE:
6  * This function tests a connect handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_connect_handle(void)
14 #else
15    VST_BOOLEAN vst_connect_handle()
16 #endif
17 {
```

```
18    VST_CONNECT_HANDLE   h;
19    VST_BOOLEAN          rc = VSE_FALSE;
20    VST_ENTERPRISE_ID    EnterpriseID;
21    VST_SOCKADDR_IN      SocketAddress;
22    VST_PROGRAM_NUMBER   ProgramNumber;
23    VST_VERSION_NUMBER   VersionNumber;
24    VST_PROCEDURE_NUMBER
         ProcedureNumber;
25    VST_PROTOCOL Protocol;
26
27    /* create the handle */
28    h = VS_Connect_Create();
29    if (h != (VST_CONNECT_HANDLE) NULL)
30    {
31       /* get values from user */
32       printf("*** connect handle
         ***\n");
33       printf("Enter enterprise ID ==>
         ");
34       EnterpriseID = atol(gets(input));
35       printf("Enter Internet sin_port
         value ==> ");
36       SocketAddress.sin_port = (short)
         atoi(gets(input));;
37       printf("Enter Internet sin_family
         value ==> ");
38       SocketAddress.sin_family =
         (short)
             atoi(gets(input));
39       printf("Enter Internet sin_addr
         value ==> ");
40       SocketAddress.sin_addr =
         atol(gets(input));
41       printf("Enter program number ==>
         ");
42       ProgramNumber =
         atol(gets(input));
43       printf("Enter version number ==>
         ");
44       VersionNumber =
         atol(gets(input));
```

```
45       printf("Enter procedure number
         ==> ");
46       ProcedureNumber =
         atol(gets(input));
47       printf("Enter Protocol ==> ");
48       Protocol = atol(gets(input));
49       /* set the fields */
50       rc = VS_Connect_SetFields(h,
51          VSID_ENTERPRISE_ID,
         EnterpriseID,
52          VSID_SOCKADDR_IN,
         SocketAddress,
53          VSID_PROGRAM_NUMBER,
         ProgramNumber,
54          VSID_VERSION_NUMBER,
         VersionNumber,
55          VSID_PROCEDURE_NUMBER,
         ProcedureNumber,
56          VSID_PROTOCOL,
         Protocol,
57          VSID_ENDFIELD);
58       if (rc)
59       {
60          vst_print_connect(h);
61       }
62       VS_Connect_Destroy(h);
63    }
64    return(rc);
65 }
```


Notes

None

See Also

- vsapi(l),
- VS_Connect_Destroy(l),
- VS_Connect_GetFields(l),
- VS_Connect_SetFields(l),
- VS_Error_GetFields(l)

VS_Connect_Destroy

`VS_Connect_Destroy` deallocates a connect handle that was allocated with `VS_Connect_Create`. A connect handle is used to pass connect information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Connect_Destroy
(VST_CONNECT_HANDLE handle)

Arguments

- `handle` = Connect handle to be destroyed.

Return Values

`VS_Connect_Destroy` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADHANDLE` - Specified handle was not a command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_connect_handle
4  *
5  * PURPOSE:
6  * This function tests a connect handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
```

```
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_connect_handle(void)
14 #else
15    VST_BOOLEAN vst_connect_handle()
16 #endif
17 {
18    VST_CONNECT_HANDLE   h;
19    VST_BOOLEAN          rc = VSE_FALSE;
20    VST_ENTERPRISE_ID    EnterpriseID;
21    VST_SOCKADDR_IN      SocketAddress;
22    VST_PROGRAM_NUMBER   ProgramNumber;
23    VST_VERSION_NUMBER   VersionNumber;
24    VST_PROCEDURE_NUMBER
         ProcedureNumber;
25    VST_PROTOCOL Protocol;
26
27    /* create the handle */
28    h = VS_Connect_Create();
29    if (h != (VST_CONNECT_HANDLE) NULL)
30    {
31       /* get values from user */
32       printf("*** connect handle
         ***\n");
33       printf("Enter enterprise ID ==>
         ");
34       EnterpriseID = atol(gets(input));
35       printf("Enter Internet sin_port
         value ==> ");
36       SocketAddress.sin_port = (short)
         atoi(gets(input));;
37       printf("Enter Internet sin_family
         value ==> ");
38       SocketAddress.sin_family =
         (short)
             atoi(gets(input));
39       printf("Enter Internet sin_addr
         value ==> ");
40       SocketAddress.sin_addr =
         atol(gets(input));
41       printf("Enter program number ==>
         ");
```

```
42       ProgramNumber =
         atol(gets(input));
43       printf("Enter version number ==>
         ");
44       VersionNumber =
         atol(gets(input));
45       printf("Enter procedure number
         ==> ");
46       ProcedureNumber =
         atol(gets(input));
47       printf("Enter Protocol ==> ");
48       Protocol = atol(gets(input));
49       /* set the fields */
50       rc = VS_Connect_SetFields(h,
51          VSID_ENTERPRISE_ID,
         EnterpriseID,
52          VSID_SOCKADDR_IN,
         SocketAddress,
53          VSID_PROGRAM_NUMBER,
         ProgramNumber,
54          VSID_VERSION_NUMBER,
         VersionNumber,
55          VSID_PROCEDURE_NUMBER,
         ProcedureNumber,
56          VSID_PROTOCOL,
         Protocol,
57          VSID_ENDFIELD);
58       if (rc)
59       {
60          vst_print_connect(h);
61       }
62       VS_Connect_Destroy(h);
63    }
64    return(rc);
65 }
```

Notes

After `VS_Connect_Destroy` has been called for a connect handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),

- VS_Connect_Create(l),
- VS_Connect_GetFields(l),
- VS_Connect_SetFields(l),
- VS_Error_GetFields(l)

VS_Connect_GetFields

`VS_Connect_GetFields` retrieves information associated with a connect handle. A connect handle is used to pass connect information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Connect_GetFields
(VST_CONNECT_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Connect handle where information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID *)	Pointer to the enterprise identifier associated with this client.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER *)	Pointer to the RPC procedure number of the client process to receive status messages from VolServ.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER *)	Pointer to the RPC program number of the client process to receive status messages.

Parameter Type	Description
VSID_PROTOCOL (VST_PROTOCOL *)	Pointer to the Internet protocol to use to return status messages to the connected client process. Valid `VSID_PROTOCOL` values are enumerated in the *vs_types.h* file.
VSID_SOCKADDR_IN (VST_SOCKADDR_IN *)	Pointer to the Internet socket address for the client process.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER *)	Pointer to the RPC version number of the client process to receive status messages.

Return Values

`VS_Connect_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_connect
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a connect handle.
8  *
9  * PARAMETERS:
```

```
10 * h : the connect handle to print
11 *
12 ************************************
        *********/
13 #ifdef ANSI_C
14    void
         vst_print_connect(VST_CONNECT_HAN
         DLE h)
15 #else
16    void vst_print_connect(h)
17    VST_CONNECT_HANDLE h;
18 #endif
19 {
20    VST_ENTERPRISE_ID
         EnterpriseID;
21    VST_SOCKADDR_IN
         SocketAddress;
22    VST_PROGRAM_NUMBER
         ProgramNumber;
23    VST_VERSION_NUMBER
         VersionNumber;
24    VST_PROCEDURE_NUMBER
         ProcedureNumber;
25    VST_PROTOCOL            Protocol;
26
27    VS_Connect_GetFields(h,
28          VSID_ENTERPRISE_ID,
         &EnterpriseID,
29          VSID_SOCKADDR_IN,
         &SocketAddress,
30          VSID_PROGRAM_NUMBER,
         &ProgramNumber,
31          VSID_VERSION_NUMBER,
         &VersionNumber,
32          VSID_PROCEDURE_NUMBER,
         &ProcedureNumber,
33          VSID_PROTOCOL,
         &Protocol,
34          VSID_ENDFIELD);
35
36    printf("******* Connect Handle
         *******\n");
```

```
37    printf("Enterprise ID = %d\n",
         EnterpriseID);
38    printf("Socket Family = %d\n",
         SocketAddress.sin_family);
39    printf("Socket Port = %d\n",
         SocketAddress.sin_port);
40    printf("Socket Address = %lu\n",
         SocketAddress.sin_addr);
41    printf("Program Number = %lu\n",
         ProgramNumber);
42    printf("Version Number = %lu\n",
         VersionNumber);
43    printf("Procedure Number = %lu\n",
         ProcedureNumber);
44    printf("Protocol = %d\n", Protocol);
45 }
```


Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Connect_Create(l),
- VS_Connect_Destroy(l),
- VS_Connect_SetFields(l),
- VS_Error_GetFields(l),
- VSCMD_ConnectQuery(l),
- VSCMD_Connect(l),
- VSCMD_Disconnect(l)

VS_Connect_SetFields

`VS_Connect_SetFields` sets the value of one or more fields in a connect handle. A connect handle is used to pass connect information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Connect_SetFields
(VST_CONNECT_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Connect handle where information is stored.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Enterprise identifier to associate with this client.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	RPC procedure number of the client process to receive status messages from VolServ.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	RPC program number of the client process to receive status messages.

Parameter Type	Description
VSID_PROTOCOL (VST_PROTOCOL)	Internet protocol to use to transmit status messages to this client. Valid VSID_PROTOCOL values are enumerated in the *vs_types.h* file.
VSID_SOCKADDR_IN (VST_SOCKADDR_IN)	Internet socket address for this client.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	RPC version number of the client process to receive status messages.

Return Values

VS_Connect_SetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a connect handle.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_connect_handle
4  *
5  * PURPOSE:
6  * This function tests a connect handle.
```

```
7  *
8  * PARAMETERS:
9  * none
10 *
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_connect_handle(void)
14 #else
15    VST_BOOLEAN vst_connect_handle()
16 #endif
17 {
18    VST_CONNECT_HANDLE      h;
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_ENTERPRISE_ID
         EnterpriseID;
21    VST_SOCKADDR_IN
         SocketAddress;
22    VST_PROGRAM_NUMBER
         ProgramNumber;
23    VST_VERSION_NUMBER
         VersionNumber;
24    VST_PROCEDURE_NUMBER
         ProcedureNumber;
25    VST_PROTOCOL Protocol;
26
27    /* create the handle */
28    h = VS_Connect_Create();
29    if (h != (VST_CONNECT_HANDLE) NULL)
30    {
31       /* get values from user */
32       printf("*** connect handle
         ***\n");
33       printf("Enter enterprise ID ==>
         ");
34       EnterpriseID = atol(gets(input));
35       printf("Enter Internet sin_port
         value ==> ");
36       SocketAddress.sin_port = (short)
         atoi(gets(input));;
```

```
37       printf("Enter Internet sin_family
         value ==> ");
38       SocketAddress.sin_family =
         (short)
             atoi(gets(input));
39       printf("Enter Internet sin_addr
         value ==> ");
40       SocketAddress.sin_addr =
         atol(gets(input));
41       printf("Enter program number ==>
         ");
42       ProgramNumber =
         atol(gets(input));
43       printf("Enter version number ==>
         ");
44       VersionNumber =
         atol(gets(input));
45       printf("Enter procedure number
         ==> ");
46       ProcedureNumber =
         atol(gets(input));
47       printf("Enter Protocol ==> ");
48       Protocol = atol(gets(input));
49       /* set the fields */
50       rc = VS_Connect_SetFields(h,
51          VSID_ENTERPRISE_ID,
         EnterpriseID,
52          VSID_SOCKADDR_IN,
         SocketAddress,
53          VSID_PROGRAM_NUMBER,
         ProgramNumber,
54          VSID_VERSION_NUMBER,
         VersionNumber,
55          VSID_PROCEDURE_NUMBER,
         ProcedureNumber,
56          VSID_PROTOCOL,
         Protocol,
57          VSID_ENDFIELD);
58       if (rc)
59       {
60          vst_print_connect(h);
61       }
```

```
62        VS_Connect_Destroy(h);
63     }
64     return(rc);
65 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Connect_Create(l),
- VS_Connect_Destroy(l),
- VS_Connect_GetFields(l),
- VS_Error_GetFields(l),
- VSCMD_Connect(l),
- VSCMD_ConnectQuery(l),
- VSCMD_Disconnect(l)

VS_Criteria_Create

`VS_Criteria_Create` allocates a VolServ API criteria handle. A criteria handle is used to pass criteria information to and from VolServ.

A criteria has three parts: the field number corresponding to a media statistic, a sort order (ascending/descending), and a group of expression handles.

Together, the group of expression commands form a single criteria to test against media that have the given field number. The criteria group handle uses criteria handles to build a comparison function that uses more than one field numbers.

Synopsis

```
VST_CRITERIA_HANDLE VS_Criteria_Create
( void )
```

Arguments

None

Return Values

`VS_Criteria_Create` returns:

- A criteria handle, if one can be allocated
- NULL, if a criteria handle cannot be allocated. An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_criteria
4  *
5  * PURPOSE:
6  * This function creates the mount
       criteria group
7  * handle and sets the values in it
       according to user
8  * input.
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria(void)
15 #else
16    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria()
17 #endif
18 {
19    int                        i;
20    int                        j;
21    int                        numcrit;
22    int                        numexpr;
23    VST_BOOLEAN                rc =
         VSE_TRUE;
24    VST_EXPRESSION_HANDLE      exprh;
25    VST_CRITERIA_HANDLE
         criteriah;
26    VST_CRITERIAGROUP_HANDLE   grouph;
27    VST_COUNT                  field;
28    VST_MOUNT_CRITERIA_ORDER   sort;
29    VST_MEDIA_STAT_VALUE       value;
30    VST_MOUNT_CRITERIA_OPT     relopt;
31    VST_CONNECTIVE_OP          conop;
32
33    /* create the criteria group */
34    grouph = VS_CriteriaGroup_Create();
```

```
35
36    if ( grouph ==
         (VST_CRITERIAGROUP_HANDLE) NULL )
37    {
38       /* out of memory -- return */
39       return (
         (VST_CRITERIAGROUP_HANDLE) NULL
         );
40    }
41
42    /* populate the criteria group with
         criteria (upto 5) */
43    printf ( “Enter number of Criteria in
         group ==> “ );
44    numcrit = atoi(gets(input));
45
46    for ( i = 0 ; i < numcrit ; i++ )
47    {
48       /* create the criteria for a media
         */
49       /* stat field */
50       criteriah = VS_Criteria_Create();
51
52       if ( criteriah ==
         (VST_CRITERIA_HANDLE) NULL )
53       {
54          /* could not allocate handle */
55          rc = VSE_FALSE;
56          break;
57       }
58
59       printf ( “Enter the media’s field
         number ==> “ );
60       field = atoi(gets(input));
61
62       printf ( “Enter the sort order
         (Ascending - 1, Descending - 2)
         ==> “ );
63       sort = atoi(gets(input));
64
65       /* set the criteria parameters */
```

```
66       VS_Criteria_SetFields (
         criteriah,
67                   VSID_FIELD, field,
68
         VSID_MOUNT_CRITERIA_ORDER, sort
69                   VSID_ENDFIELD );
70
71       /* populate the critera with
         expressions */
72       /* (up to 4) */
73       printf ( “Enter the number of
         criteria expressions ==> “ );
74       numexpr = atoi(gets(input));
75
76       for ( j = 0 ; j < numexpr ; j++ )
77       {
78       /* create an expression for this
         criteria */
79          exprh =
         VS_Expression_Create();
80
81          if ( exprh ==
         (VST_EXPRESSION_HANDLE) NULL )
82          {
83          /* could not allocate memory
         for this handle */
84             rc = VSE_FALSE;
85             break;
86          }
87
88          printf ( “Enter relational
         option (eq 1, gt 2, ge 3, lt 4, le
         5, ne 6) ==> “ );
89          relopt = atoi(gets(input));
90
91          printf ( “Enter the media field
         value ==> “ );
92          gets( value);
93
94          printf ( “Enter connective
         operation (none 0, and 1, or 2)
         ==> “ );
```

```
95          conop = atoi(gets(input));
96
97          /* set the expression's
          parameters */
98          VS_Expression_SetFields (
          exprh,
99                VSID_MOUNT_CRITERIA_OPT,
          relopt,
100               VSID_CONNECTIVE_OP,
          conop,
101               VSID_MEDIA_STAT_VALUE,
          value,
102               VSID_ENDFIELD );
103
104         /* add the expression to the
          criteria */
105         VS_Criteria_SetFields (
          criteriah,
106
          VSID_EXPRESSION_HANDLE_ENTRY, j,
          exprh,
107               VSID_ENDFIELD );
108       }
109
110       /* add the criteria to the
          criteria group */
111       VS_CriteriaGroup_SetFields (
          grouph,
112               VSID_CRITERIA_HANDLE_ENTRY,
          i, criteriah,
113               VSID_ENDFIELD );
114   }
115
116   /* if it failed, destroy the criteria
          group handle */
117   if ( rc == VSE_FALSE )
118   {
119       /* criteria group will destroy any
          */
120       /* criteria and their expressions
          */
121       /* for us */
```

```
122      VS_CriteriaGroup_Destroy ( grouph
         );
123
124      grouph =
         (VST_CRITERIAGROUP_HANDLE) NULL;
125   }
126
127   return ( grouph );
128}
```

Notes

None

See Also

- vsapi(l),
- VS_Criteria_Delete(l),
- VS_Criteria_GetFields(l),
- VS_Criteria_SetFields(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_GetFields(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Error_GetFields(l),
- VS_Expression_Create(l),
- VS_Expression_Delete(l),
- VS_Expression_GetFields(l),
- VS_Expression_SetFields(l),
- VSCMD_Mount(l)

VS_Criteria_Destroy

VS_Criteria_Destroy deallocates a criteria handle that was allocated with VS_Criteria_Create. A criteria handle is used to pass criteria information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Criteria_Destroy
(VST_CRITERIA_HANDLE handle)

Arguments

- handle = Criteria handle to be destroyed.

Return Values

VS_Criteria_Destroy returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_criteria
4  *
5  * PURPOSE:
6  * This function creates the mount
       criteria group
7  * handle and sets the values in it
       according to user
8  *input.
9  *
10 * PARAMETERS:
11 * none
```

```
12 *
13 ************************************
       *********/
14 #ifdef ANSI_C
15    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria(void)
16 #else
17    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria()
18 #endif
19 {
20    int                        i;
21    int                        j;
22    int                        numcrit;
23    int                        numexpr;
24    VST_BOOLEAN                rc =
         VSE_TRUE;
25    VST_EXPRESSION_HANDLE      exprh;
26    VST_CRITERIA_HANDLE
         criteriah;
27    VST_CRITERIAGROUP_HANDLE   grouph;
28    VST_COUNT                  field;
29    VST_MOUNT_CRITERIA_ORDER   sort;
30    VST_MEDIA_STAT_VALUE       value;
31    VST_MOUNT_CRITERIA_OPT     relopt;
32    VST_CONNECTIVE_OP          conop;
33
34    /* create the criteria group */
35    grouph = VS_CriteriaGroup_Create();
36    if ( grouph ==
         (VST_CRITERIAGROUP_HANDLE) NULL )
37    {
38       /* out of memory -- return */
39       return (
         (VST_CRITERIAGROUP_HANDLE) NULL
         );
40    }
41    /* populate the criteria group with
         criteria */
42    /* (upto 5) */
43    printf ( “Enter number of Criteria in
         group ==> “ );
```

```
44    numcrit = atoi(gets(input));
45    for ( i = 0 ; i < numcrit ; i++ )
46    {
47       /* create the criteria for a media
      stat field */
48       criteriah = VS_Criteria_Create();
49       if ( criteriah ==
      (VST_CRITERIA_HANDLE) NULL )
50       {
51          /* could not allocate handle */
52          rc = VSE_FALSE;
53          break;
54       }
55       printf ( “Enter the media’s field
      number ==> “ );
56       field = atoi(gets(input));
57       printf ( “Enter the sort order
      (Ascending - 1, Descending - 2)
      ==>“ );
58       sort = atoi(gets(input));
59       /* set the criteria parameters */
60       VS_Criteria_SetFields (
      criteriah,
61             VSID_FIELD,
      field,
62             VSID_MOUNT_CRITERIA_ORDER,
      sort,
63             VSID_ENDFIELD );
64       /* populate the critera with
      expressions */
65       /* (upto 4) */
66       printf ( “Enter the number of
      criteria expressions ==> “ );
67       numexpr = atoi(gets(input));
68       for ( j = 0 ; j < numexpr ; j++ )
69       {
70          /* create an expression for
      this criteria */
71          exprh =
      VS_Expression_Create();
72          if ( exprh ==
      (VST_EXPRESSION_HANDLE) NULL )
```

```
73          {
74             /* could not allocate memory
            for this */
75             /* handle */
76             rc = VSE_FALSE;
77             break;
78          }
79          printf ( “Enter relational
            option (eq 1, gt 2, ge 3, lt 4, le
            5, ne 6) ==> “ );
80          relopt = atoi(gets(input));
81          printf ( “Enter the media field
            value ==> “ );
82          gets( value);
83          printf ( “Enter connective
            operation (none 0, and 1, or 2)
            ==> “ );
84          conop = atoi(gets(input));
85          /* set the expression’s
            parameters */
86          VS_Expression_SetFields (
            exprh,
87             VSID_MOUNT_CRITERIA_OPT,
            relopt,
88             VSID_CONNECTIVE_OP,
            conop,
89             VSID_MEDIA_STAT_VALUE,
            value,
90             VSID_ENDFIELD );
91          /* add the expression to the
            criteria */
92          VS_Criteria_SetFields (
            criteriah,
93
            VSID_EXPRESSION_HANDLE_ENTRY, j,
            exprh,
94             VSID_ENDFIELD );
95       }
96       /* add the criteria to the
            criteria group */
97       VS_CriteriaGroup_SetFields (
            grouph,
```

```
98             VSID_CRITERIA_HANDLE_ENTRY,
       i, criteriah,
99             VSID_ENDFIELD );
100   }
101   /* if it failed, destroy the criteria
       group handle */
102   if ( rc == VSE_FALSE )
103   {
104      /* criteria group will destroy any
       */
105      /* criteria and their expressions
       */
106      /* for us, so the only thing that
       is really */
107      /* needed here is a call to*/
108      /* VS_CriteriaGroup_Destroy. This
       is written*/
109      /* out the 'long way' for
       documentation*/
110      /* purposes. First, get the number
       of criteria */
111
         VS_CriteriaGroup_GetFields(grouph
         ,
112            VSID_NUMBER_ENTRIES,
         &numcrit,
113            VSID_ENDFIELD);
114      for (i = 0; i < numcrit; i++)
115      {
116         /* get a criteria handle */
117
         VS_CriteriaGroup_GetFields(grouph
         ,
118            VSID_CRITERIA_HANDLE_ENTRY,
         i, &criteriah,
119            VSID_ENDFIELD);
120         /* get the number of
         expressions */
121
         VS_Criteria_GetFields(criteriah,
122            VSID_NUMBER_ENTRIES,
         &numexpr,
```

```
123            VSID_ENDFIELD);
124         for (j = 0; j < numexpr; j++)
125         {
126            /* get the expressions from
           the criteria */
127
           VS_Criteria_GetFields(criteriah,
128
           VSID_EXPRESSION_HANDLE_ENTRY, j,
           &exprh,
129              VSID_ENDFIELD);
130            /* destroy the expression
           handle */
131            /*
           VS_Expression_Destroy(exprh);*/
132            /* let criteria handle know
           that the */
133            /* expression handle has
           been destroyed */
134
           VS_Criteria_SetFields(criteriah,
135
           VSID_EXPRESSION_HANDLE_ENTRY, j,
           NULL,
136              VSID_ENDFIELD);
137         }
138         /* now, destroy Criteria
           handle. */
139
           VS_Criteria_Destroy(criteriah);
140         /* let the criteria group
           handle know */
141         /* that Criteria handle has
           been */
142         /* destroyed. */
143
           VS_CriteriaGroup_SetFields(grouph
           ,
144
           VSID_CRITERIA_HANDLE_ENTRY, i,
           NULL,
145              VSID_ENDFIELD);
```

```
146      }
147      /* finally, destroy the criteria
         group handle. */
148      VS_CriteriaGroup_Destroy ( grouph
         );
149      grouph =
         (VST_CRITERIAGROUP_HANDLE) NULL;
150   }
151   return ( grouph );
152}
```

Notes

After `VS_Criteria_Destroy` has been called for a criteria handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Criteria_Create(l),
- VS_Criteria_GetFields(l),
- VS_Criteria_SetFields(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_GetFields(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Error_GetFields(l),
- VS_Expression_Create,(l)
- VS_Expression_Delete(l),
- VS_Expression_GetFields(l),
- VS_Expression_SetFields(l),
- VSCMD_Mount(l)

VS_Criteria_GetFields

`VS_Criteria_GetFields` retrieves information associated with a criteria handle. A criteria group handle is used to pass criteria information to and from VolServ.

Synopsis

```
VST_BOOLEAN VS_Criteria_GetFields
( VST_CRITERIA_HANDLE handle,
"…",
VSID_ENDFIELD )
```

Arguments

- `handle` = Criteria handle where information is stored.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_NUMBER_ENTRIES (VST_COUNT *)	Pointer to the number of expression handles within this criteria handle.
VSID_FIELD (VST_COUNT *)	Pointer to the field number of the media statistic for this criteria.
VSID_MOUNT_CRITERIA_ORDER (VST_MOUNT_CRITERIA_ORDER*)	Pointer to the sort ordering for the media that pass the given criteria expressions.
VSID_EXPRESSION_HANDLE_ENTRY (int)	Expression handle and its place in the criteria.

Parameter Type	Description
(VST_EXPRESSION_HANDLE *)	Pointer to the expression handle and its place in the criteria.

Return Values

`VS_Criteria_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a criteria handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_criteria_group
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a criteria group handle.
8  *
9  * PARAMETERS:
10 * grouph : the mount handle to print
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
```

```
14    void
          vst_print_criteria_group
         (VST_CRITERIAGROUP_HANDLE grouph)
15 #else
16    void
         vst_print_criteria_group(grouph)
17    VST_CRITERIAGROUP_HANDLE grouph;
18 #endif
19 {
20    int                          i, j;
21    int                          numcrit =
         0;
22    int                          numexpr =
         0;
23
24    VST_EXPRESSION_HANDLE
                exprh =
         (VST_EXPRESSION_HANDLE) NULL;
25    VST_CRITERIA_HANDLE
                criteriah =
         (VST_CRITERIA_HANDLE) NULL;
26
27    VST_COUNT                    field;
28    VST_MOUNT_CRITERIA_ORDER     sort;
29    VST_MEDIA_STAT_VALUE         value;
30    VST_MOUNT_CRITERIA_OPT       relopt;
31    VST_CONNECTIVE_OP            conop;
32
33    /* get the number of criteria within
         this group */
34    VS_CriteriaGroup_GetFields ( grouph,
35       VSID_NUMBER_ENTRIES, &numcrit,
         VSID_ENDFIELD );
36
37    for ( i = 0 ; i < numcrit ; i++ )
38    {
39       /* get the criteria to print */
40       VS_CriteriaGroup_GetFields (
         grouph,
41          VSID_CRITERIA_HANDLE_ENTRY, i,
         &criteriah,
42          VSID_ENDFIELD );
```

```
43
44        /* get the criteria parameters */
45        VS_Criteria_GetFields (
          criteriah,
46           VSID_FIELD,
          &field,
47           VSID_MOUNT_CRITERIA_ORDER,
          &sort,
48           VSID_NUMBER_ENTRIES,
          &numexpr,
49           VSID_ENDFIELD );
50
51        printf ( "*** Criteria # %d
          ***\n", i );
52        printf ( " Field Index ==> %d\n",
          field );
53        printf ( " Mount Criteria Order
          ==> %d\n", sort );
54        printf ( "Number of Expressions
          ==> %d\n", numexpr );
55
56        for ( j = 0 ; j < numexpr ; j++ )
57        {
58           /* get the expression to print
          */
59           VS_Criteria_GetFields (
          criteriah,
60
          VSID_EXPRESSION_HANDLE_ENTRY, j,
          &exprh,
61              VSID_ENDFIELD );
62
63           /* get the expression's
          parameters */
64           VS_Expression_GetFields (
          exprh,
65              VSID_MOUNT_CRITERIA_OPT,
          &relopt,
66              VSID_CONNECTIVE_OP,
          &conop,
67              VSID_MEDIA_STAT_VALUE,
          value,
```

```
68              VSID_ENDFIELD );
69
70           printf ( "*** Expression # %d
          ***\n", j );
71           printf ( "Mount Criteria
          Option ==> %d\n", relopt);
72           printf ( " Media Stat Value ==>
          %s\n", value );
73           printf ( " Connective
          Operation ==> %d\n", conop );
74           }
75      }
76
77      return;
78  }
```

Notes

The `VSID_EXPRESSION_HANDLE_ENTRY` parameter requires that two arguments be passed instead of one.

- The first argument is the entry number in the criteria.
- The second argument is Pointer to the location where the value is stored.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Criteria_Create(l),
- VS_Criteria_Destroy(l),
- VS_Criteria_SetFields(l),
- VS_Error_Getfields(l),
- VS_Expression_Create(l),

- VS_Expression_Destroy(l),
- VS_Expression_GetFields (l),
- VS_Expression_SetFields(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_GetFields(l),
- VS_CriteriaGroup_SetFields(l),
- VSCMD_Mount(l)

VS_Criteria_SetFields

`VS_Criteria_SetFields` sets the value of one or more field in a criteria handle. A criteria handle is used to pass criteria information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Criteria_SetFields
(VST_CRITERIA_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Criteria handle where information is stored.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD = Re`quired at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_FIELD (VST_COUNT)	Field number of the media statistic for this criteria.
VSID_MOUNT_CRITERIA_ORDER (VST_MOUNT_CRITERIA_ORDER)	Sort ordering for the media that pass the given criteria expressions.
VSID_EXPRESSION_HANDLE_ENTRY (int)	Expression handle and its place in the criteria.
(VST_EXPRESSION_HANDLE)	Expression handle for the criteria.

Return Values

VS_Criteria_SetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a criteria handle.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_criteria
4  *
5  * PURPOSE:
6  * This function creates the mount
       criteria group
7  * handle and sets the values in it
       according to
8  * user input.
9  *
10 * PARAMETERS:
11 * none
12 *
13 *************************************
       *********/
14 #ifdef ANSI_C
```

```
15    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria(void)
16 #else
17    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria()
18 #endif
19 {
20    int                        i;
21    int                        j;
22    int                        numcrit;
23    int                        numexpr;
24    VST_BOOLEAN                rc =
         VSE_TRUE;
25    VST_EXPRESSION_HANDLE      exprh;
26    VST_CRITERIA_HANDLE
         criteriah;
27    VST_CRITERIAGROUP_HANDLE   grouph;
28    VST_COUNT                  field;
29    VST_MOUNT_CRITERIA_ORDER   sort;
30    VST_MEDIA_STAT_VALUE       value;
31    VST_MOUNT_CRITERIA_OPT     relopt;
32    VST_CONNECTIVE_OP          conop;
33
34    /* create the criteria group */
35    grouph = VS_CriteriaGroup_Create();
36
37    if ( grouph ==
         (VST_CRITERIAGROUP_HANDLE) NULL )
38    {
39       /* out of memory -- return */
40       return (
         (VST_CRITERIAGROUP_HANDLE) NULL
         );
41    }
42
43    /* populate the criteria group with
         criteria */
44    /* (upto 5) */
45    printf ( “Enter number of Criteria in
         group ==> “ );
46    numcrit = atoi(gets(input));
47
```

```
48    for ( i = 0 ; i < numcrit ; i++ )
49    {
50       /* create the criteria for a media
         stat field */
51       criteriah = VS_Criteria_Create();
52
53       if ( criteriah ==
         (VST_CRITERIA_HANDLE) NULL )
54       {
55          /* could not allocate handle */
56          rc = VSE_FALSE;
57          break;
58       }
59
60       printf ( "Enter the media's field
         number ==> " );
61       field = atoi(gets(input));
62
63       printf ( "Enter the sort order
         (Ascending - 1, Descending - 2)
         ==> " );
64       sort = atoi(gets(input));
65
66       /* set the criteria parameters */
67       VS_Criteria_SetFields (
         criteriah,
68                   VSID_FIELD, field,
69
         VSID_MOUNT_CRITERIA_ORDER, sort,
70                   VSID_ENDFIELD );
71       /* populate the critera with
         expressions */
72       /* (upto 4) */
73       printf ( "Enter the number of
         criteria expressions ==> " );
74       numexpr = atoi(gets(input));
75
76       for ( j = 0 ; j < numexpr ; j++ )
77       {
78          /* create an expression for
         this criteria */
```

```
79          exprh =
        VS_Expression_Create();
80
81          if ( exprh ==
        (VST_EXPRESSION_HANDLE) NULL )
82          {
83             /* could not allocate memory
        for this */
84             /* handle */
85             rc = VSE_FALSE;
86             break;
87          }
88
89          printf ( “Enter relational
        option (eq 1, gt 2, ge 3, lt 4, le
        5, ne 6) ==> “ );
90          relopt = atoi(gets(input));
91
92          printf ( “Enter the media field
        value ==> “ );
93          gets( value);
94
95          printf ( “Enter connective
        operation (none 0, and 1, or 2)
        ==> “ );
96          conop = atoi(gets(input));
97
98          /* set the expression’s
        parameters */
99          VS_Expression_SetFields (
        exprh,
100                VSID_MOUNT_CRITERIA_OPT,
        relopt,
101                VSID_CONNECTIVE_OP,
        conop,
102                VSID_MEDIA_STAT_VALUE,
        value,
103                VSID_ENDFIELD );
104
105         /* add the expression to the
        criteria */
```

```
106         VS_Criteria_SetFields (
          criteriah,
107
          VSID_EXPRESSION_HANDLE_ENTRY, j,
          exprh,
108               VSID_ENDFIELD );
109       }
110
111       /* add the criteria to the
          criteria group */
112       VS_CriteriaGroup_SetFields (
          grouph,
113          VSID_CRITERIA_HANDLE_ENTRY, i,
          criteriah,
114          VSID_ENDFIELD );
115    }
116
117    /* if it failed, destroy the criteria
          group handle */
118    if ( rc == VSE_FALSE )
119    {
120       /* criteria group will destroy any
          */
121       /* criteria and their expressions
          */
122       /* for us */
123       VS_CriteriaGroup_Destroy ( grouph
          );
124
125       grouph =
          (VST_CRITERIAGROUP_HANDLE) NULL;
126    }
127
128    return ( grouph );
129}
```

Notes

The `VSID_EXPRESSION_HANDLE_ENTRY` parameter requires that two arguments be passed instead of one.

- The first argument is the entry number in the criteria.
- The second argument is Pointer to the location where the value is stored.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Criteria_Create(l),
- VS_Criteria_Destroy(l),
- VS_Criteria_GetFields(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_GetFields(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Error_GetFields(l),
- VS_Expression_Create(l),
- VS_Expression_Destroy(l),
- VS_Expression_GetFields(l),
- VS_Expression_SetFields(l),
- VSCMD_Mount(l)

VS_CriteriaGroup_Create

`VS_CriteriaGroup_Create` allocates a VolServ API criteria group handle. A criteria group handle is used to pass criteria group information to and from VolServ.

Synopsis

```
VST_CRITERIAGROUP_HANDLE
VS_CriteriaGroup_Create
( void )
```

Arguments

None

Return Values

`VS_CriteriaGroup_Create` returns:

- A criteria group handle, if one can be allocated.
- NULL, if a criteria group handle cannot be allocated. An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_criteria
4  *
5  * PURPOSE:
6  * This function creates the mount
       criteria group
7  * handle and sets the values in it
       according to user
8  * input.
9  *
10 * PARAMETERS:
11 * none
12 *
```

```
13 ************************************
      *********/
14 #ifdef ANSI_C
15    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria(void)
16 #else
17    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria()
18 #endif
19 {
20    int                        i;
21    int                        j;
22    int                        numcrit;
23    int                        numexpr;
24    VST_BOOLEAN                rc =
         VSE_TRUE;
25    VST_EXPRESSION_HANDLE      exprh;
26    VST_CRITERIA_HANDLE
         criteriah;
27    VST_CRITERIAGROUP_HANDLE   grouph;
28    VST_COUNT                  field;
29    VST_MOUNT_CRITERIA_ORDER   sort;
30    VST_MEDIA_STAT_VALUE       value;
31    VST_MOUNT_CRITERIA_OPT     relopt;
32    VST_CONNECTIVE_OP          conop;
33
34    /* create the criteria group */
35    grouph = VS_CriteriaGroup_Create();
36
37    if ( grouph ==
         (VST_CRITERIAGROUP_HANDLE) NULL )
38    {
39       /* out of memory -- return */
40       return (
         (VST_CRITERIAGROUP_HANDLE) NULL
         );
41    }
42
43    /* populate the criteria group with
         criteria */
44    /* (upto 5) */
```

```
45    printf ( “Enter number of Criteria in
         group ==> “ );
46    numcrit = atoi(gets(input));
47
48    for ( i = 0 ; i < numcrit ; i++ )
49    {
50       /* create the criteria for a media
         stat field */
51       criteriah = VS_Criteria_Create();
52
53       if ( criteriah ==
         (VST_CRITERIA_HANDLE) NULL )
54       {
55          /* could not allocate handle */
56          rc = VSE_FALSE;
57          break;
58       }
59
60       printf ( “Enter the media’s field
         number ==> “ );
61       field = atoi(gets(input));
62
63       printf ( “Enter the sort order
         (Ascending - 1, Descending - 2)
         ==> “ );
64       sort = atoi(gets(input));
65
66       /* set the criteria parameters */
67       /VS_Criteria_SetFields (
         criteriah,
68                   VSID_FIELD,
         field,
69
         VSID_MOUNT_CRITERIA_ORDER, sort,
70                   VSID_ENDFIELD );
71
72       /* populate the critera with
         expressions */
73       /* (upto 4) */
74       printf ( “Enter the number of
         criteria expressions ==> “ );
75       numexpr = atoi(gets(input));
```

```
76
77       for ( j = 0 ; j < numexpr ; j++ )
78       {
79          /* create an expression for
         this criteria */
80          exprh =
         VS_Expression_Create();
81
82          if ( exprh ==
         (VST_EXPRESSION_HANDLE) NULL )
83          {
84             /* could not allocate memory
         for this */
85             /* handle */
86             rc = VSE_FALSE;
87             break;
88          }
89
90          printf ( “Enter relational
         option (eq 1, gt 2, ge 3, lt 4, le
         5, ne 6) ==> “ );
91          relopt = atoi(gets(input));
92
93          printf ( “Enter the media field
         value ==> “ );
94          gets( value);
95
96          printf ( “Enter connective
         operation (none 0, and 1, or 2)
         ==> “ );
97          conop = atoi(gets(input));
98
99          /* set the expression’s
         parameters */
100         VS_Expression_SetFields (
         exprh,
101            VSID_MOUNT_CRITERIA_OPT,
         relopt,
102            VSID_CONNECTIVE_OP,
         conop,
103            VSID_MEDIA_STAT_VALUE,
         value,
```

```
104            VSID_ENDFIELD );
105
106         /* add the expression to the
            criteria */
107         VS_Criteria_SetFields (
            criteriah,
108
            VSID_EXPRESSION_HANDLE_ENTRY, j,
            exprh,
109            VSID_ENDFIELD );
110
111      }
112      /* add the criteria to the
            criteria group */
113      VS_CriteriaGroup_SetFields (
            grouph,
114         VSID_CRITERIA_HANDLE_ENTRY, i,
            criteriah,
115         VSID_ENDFIELD );
116   }
117
118   /* if it failed, destroy the criteria
         group handle */
119   if ( rc == VSE_FALSE )
120   {
121      /* criteria group will destroy any
         */
122      /* criteria and their expressions
         */
123      /* for us */
124      VS_CriteriaGroup_Destroy ( grouph
         );
125
126      grouph =
         (VST_CRITERIAGROUP_HANDLE) NULL;
127   }
128
129   return ( grouph );
130}
```

Notes

A criteria group can hold up to five criteria handles.

A criteria group is used by the Mount and MultiMount commands when a client specifies criteria to be used by VolServ when selecting the medium or media to honor the Mount/MultiMount request. Criteria groups applicable to a Mount/MultiMount request can be specified either on the command itself or in a Mount handle.

See Also

- vsapi(l),
- VS_Criteria_Create(l),
- VS_Criteria_Destroy(l),
- VS_Criteria_GetFields(l),
- VS_Criteria_SetFields(l), V
- S_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_GetFields(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Error_GetFields(l),
- VS_Expression_Create(l),
- VS_Expression_Destroy(l),
- VS_Expression_GetFields(l),
- VS_Expression_SetFields(l),
- VS_Mount_SetFields(l),
- VSCMD_Mount(l)

VS_ CriteriaGroup _Destroy

`VS_CriteriaGroup_Destroy` deallocates a criteria group handle that was allocated with `VS_CriteriaGroup_Create`. A criteria group handle is used to pass criteria group information to and from VolServ.

Synopsis

VST_BOOLEAN VS_CriteriaGroup_Destroy
(VST_CRITERIAGROUP_HANDLE handle)

Arguments

- `handle =` Criteria group handle to be destroyed.

Return Values

`VS_CriteriaGroup_Destroy` returns a criteria group handle if one can be allocated.

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - The appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADHANDLE` - Specified handle was not a criteria group handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_criteria
4  *
5  * PURPOSE:
6  * This function creates the mount
       criteria group
7  * handle and sets the values in it
       according to
8  * user input.
```

```
 *
 * PARAMETERS:
 * none
 *
 ************************************
     *********/
#ifdef ANSI_C
   VST_CRITERIAGROUP_HANDLE
      vst_create_mount_criteria(void)
#else
   VST_CRITERIAGROUP_HANDLE
      vst_create_mount_criteria()
#endif
{
   int                        i;
   int                        j;
   int                        numcrit;
   int                        numexpr;
   VST_BOOLEAN                rc =
      VSE_TRUE;
   VST_EXPRESSION_HANDLE      exprh;
   VST_CRITERIA_HANDLE
      criteriah;
   VST_CRITERIAGROUP_HANDLE   grouph;
   VST_COUNT                  field;
   VST_MOUNT_CRITERIA_ORDER   sort;
   VST_MEDIA_STAT_VALUE       value;
   VST_MOUNT_CRITERIA_OPT     relopt;
   VST_CONNECTIVE_OP          conop;

   /* create the criteria group */
   grouph = VS_CriteriaGroup_Create();

   if ( grouph ==
      (VST_CRITERIAGROUP_HANDLE) NULL )
   {
      /* out of memory -- return */
      return (
      (VST_CRITERIAGROUP_HANDLE) NULL
      );
   }

```

```
43    /* populate the criteria group with
         criteria */
44    /* (upto 5) */
45    printf ( “Enter number of Criteria in
         group ==> “ );
46    numcrit = atoi(gets(input));
47
48    for ( i = 0 ; i < numcrit ; i++ )
49    {
50       /* create the criteria for a media
         stat field */
51       criteriah = VS_Criteria_Create();
52
53       if ( criteriah ==
         (VST_CRITERIA_HANDLE) NULL )
54       {
55          /* could not allocate handle */
56          rc = VSE_FALSE;
57          break;
58       }
59
60       printf ( “Enter the media’s field
         number ==> “ );
61       field = atoi(gets(input));
62
63       printf ( “Enter the sort order
         (Ascending - 1, Descending - 2)
         ==> “ );
64       sort = atoi(gets(input));
65
66       /* set the criteria parameters */
67       /VS_Criteria_SetFields (
         criteriah,
68                   VSID_FIELD,
         field,
69
         VSID_MOUNT_CRITERIA_ORDER, sort,
70                   VSID_ENDFIELD );
71
72       /* populate the critera with
         expressions */
73       /* (upto 4) */
```

```
74        printf ( “Enter the number of
          criteria expressions ==> “ );
75        numexpr = atoi(gets(input));
76
77        for ( j = 0 ; j < numexpr ; j++ )
78        {
79           /* create an expression for
          this criteria */
80           exprh =
          VS_Expression_Create();
81
82           if ( exprh ==
          (VST_EXPRESSION_HANDLE) NULL )
83           {
84              /* could not allocate memory
          for this */
85              /* handle */
86              rc = VSE_FALSE;
87              break;
88           }
89
90           printf ( “Enter relational
          option (eq 1, gt 2, ge 3, lt 4, le
          5, ne 6) ==> “ );
91           relopt = atoi(gets(input));
92
93           printf ( “Enter the media field
          value ==> “ );
94           gets( value);
95
96           printf ( “Enter connective
          operation (none 0, and 1, or 2)
          ==> “ );
97           conop = atoi(gets(input));
98
99           /* set the expression’s
          parameters */
100          VS_Expression_SetFields (
          exprh,
101             VSID_MOUNT_CRITERIA_OPT,
          relopt,
```

```
102             VSID_CONNECTIVE_OP,
        conop,
103             VSID_MEDIA_STAT_VALUE,
        value,
104             VSID_ENDFIELD );
105
106          /* add the expression to the
        criteria */
107          VS_Criteria_SetFields (
        criteriah,
108
        VSID_EXPRESSION_HANDLE_ENTRY, j,
        exprh,
109             VSID_ENDFIELD );
110
111       }
112       /* add the criteria to the
        criteria group */
113       VS_CriteriaGroup_SetFields (
        grouph,
114          VSID_CRITERIA_HANDLE_ENTRY, i,
        criteriah,
115          VSID_ENDFIELD );
116    }
117
118    /* if it failed, destroy the criteria
        group handle */
119    if ( rc == VSE_FALSE )
120    {
121       /* criteria group will destroy any
        */
122       /* criteria and their expressions
        */
123       /* for us */
124       VS_CriteriaGroup_Destroy ( grouph
        );
125
126       grouph =
        (VST_CRITERIAGROUP_HANDLE) NULL;
127    }
128
129    return ( grouph );
```

```
130}
```

Notes

After `VS_CriteriaGroup_Destroy` has been called for a criteria group handle, that handle is no longer valid and should not be used.

Destroying the CriteriaGroup handle automatically destroys the underlying criteria and expressions.

See Also

- vsapi(l),
- VS_Criteria_Create(l),
- VS_Criteria_Destroy(l),
- VS_Criteria_GetFields(l),
- VS_Criteria_SetFields(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_GetFields(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Error_GetFields(l),
- VS_Expression_Create(l),
- VS_Expression_Destroy(l),
- VS_Expression_GetFields(l),
- VS_Expression_SetFields(l),
- VSCMD_Mount(l)

VS_ CriteriaGroup _GetFields

`VS_CriteriaGroup_GetFields` retrieves information associated with a criteria group handle. A criteria group handle is used to pass criteria group information to and from VolServ.

Synopsis

VST_BOOLEAN VS_CriteriaGroup_GetFields
(VST_CRITERIAGROUP_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Media handle where information is stored.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_NUMBER_ENTRIES (VST_COUNT *)	Pointer to the number of criteria handles within this criteria group handle.
VSID_CRITERIA_HANDLE_ENTRY (int)	Criteria handle and its place in the criteria group.
(VST_CRITERIA_HANDLE *)	Pointer to Criteria handle for this group.

Return Values

`VS_CriteriaGroup_GetFields` returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - The appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a criteria group handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_criteria_group
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a criteria group handle.
8  *
9  * PARAMETERS:
10 * grouph : the mount handle to print
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    void
          vst_print_criteria_group
          (VST_CRITERIAGROUP_HANDLE grouph)
15 #else
16    void
          vst_print_criteria_group(grouph)
17    VST_CRITERIAGROUP_HANDLE grouph;
18 #endif
19 {
20    int                        i, j;
```

```
21    int                          numcrit =
         0;
22    int                          numexpr =
         0;
23
24    VST_EXPRESSION_HANDLE exprh =
         (VST_EXPRESSION_HANDLE) NULL;
25    VST_CRITERIA_HANDLE criteriah =
         (VST_CRITERIA_HANDLE) NULL;
26
27    VST_COUNT field;
28    VST_MOUNT_CRITERIA_ORDER   sort;
29    VST_MEDIA_STAT_VALUE       value;
30    VST_MOUNT_CRITERIA_OPT     relopt;
31    VST_CONNECTIVE_OP          conop;
32
33    /* get the number of criteria within
         this group */
34    VS_CriteriaGroup_GetFields ( grouph,
35       VSID_NUMBER_ENTRIES, &numcrit,
36       VSID_ENDFIELD );
37
38    for ( i = 0 ; i < numcrit ; i++ )
39  {
40       /* get the criteria to print */
41       VS_CriteriaGroup_GetFields (
         grouph,
42          VSID_CRITERIA_HANDLE_ENTRY, i,
         &criteriah,
43          VSID_ENDFIELD );
44
45       /* get the criteria parameters */
46       VS_Criteria_GetFields (
         criteriah,
47          VSID_FIELD,
         &field,
48          VSID_MOUNT_CRITERIA_ORDER,
         &sort,
49          VSID_NUMBER_ENTRIES,
         &numexpr,
50          VSID_ENDFIELD );
51
```

```
52        printf ( "*** Criteria # %d
          ***\n", i );
53        printf ( " Field Index ==> %d\n",
          field );
54        printf ( " Mount Criteria Order
          ==> %d\n", sort );
55        printf ( "Number of Expressions
          ==> %d\n", numexpr );
56
57        for ( j = 0 ; j < numexpr ; j++ )
58        {
59           /* get the expression to print
          */
60           VS_Criteria_GetFields (
          criteriah,
61
          VSID_EXPRESSION_HANDLE_ENTRY, j,
          &exprh,
62                VSID_ENDFIELD );
63
64           /* get the expression's
          parameters */
65           VS_Expression_GetFields (
          exprh,
66                VSID_MOUNT_CRITERIA_OPT,
          &relopt,
67                VSID_CONNECTIVE_OP,
          &conop,
68                VSID_MEDIA_STAT_VALUE,
          value,
69                VSID_ENDFIELD );
70
71           printf ( "*** Expression # %d
          ***\n", j );
72           printf ( "Mount Criteria
          Option ==> %d\n", relopt);
73           printf ( " Media Stat Value ==>
          %s\n", value);
74           printf ( " Connective
          Operation ==> %d\n", conop);
75        }
76     }
```

```
77
78    return;
79 }
```

Notes

The `VSID_CRITERIA_HANDLE_ENTRY` parameter requires that two arguments be passed instead of one. The first argument passed is the entry number in the criteria group table. The second argument is a pointer to the location where the value is stored.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Criteria_Create(l),
- VS_Criteria_Destroy(l),
- VS_Criteria_GetFields(l),
- VS_Criteria_SetFields(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Error_GetFields(l),
- VS_Expression_Create(l),
- VS_Expression_Destroy(l),
- VS_Expression_GetFields(l),
- VS_Expression_SetFields(l),
- VSCMD_Mount(l)

VS_CriteriaGroup_SetFields

VS_CriteriaGroup_SetFields sets the value of one or more fields in a criteria group handle. A criteria group handle is used to pass information to and from VolServ.

Synopsis

```
VST_BOOLEAN VS_CriteriaGroup_SetFields
( VST_CRITERIAGROUP_HANDLE handle,
 "…",
VSID_ENDFIELD )
```

Arguments

- handle = Criteria group handle where information is stored.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CRITERIA_HANDLE_ENTRY(int)	Criteria handle and its place in the criteria group.
(VST_CRITERIA_HANDLE)	Criteria handle for this group.

Return Values

`VS_CriteriaGroup_SetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - The appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a criteria group handle.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_criteria
4  *
5  * PURPOSE:
6  * This function creates the mount
       criteria group
7  * handle and sets the values in it
       according to user
8  * input.
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
```

```
14    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria(void)
15 #else
16    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria()
17 #endif
18 {
19    int                        i;
20    int                        j;
21    int                        numcrit;
22    int                        numexpr;
23    VST_BOOLEAN                rc =
         VSE_TRUE;
24    VST_EXPRESSION_HANDLE      exprh;
25    VST_CRITERIA_HANDLE
         criteriah;
26    VST_CRITERIAGROUP_HANDLE   grouph;
27    VST_COUNT                  field;
28    VST_MOUNT_CRITERIA_ORDER   sort;
29    VST_MEDIA_STAT_VALUE       value;
30    VST_MOUNT_CRITERIA_OPT     relopt;
31    VST_CONNECTIVE_OP          conop;
32
33    /* create the criteria group */
34    grouph = VS_CriteriaGroup_Create();
35
36    if ( grouph ==
         (VST_CRITERIAGROUP_HANDLE) NULL )
37    {
38       /* out of memory -- return */
39       return (
         (VST_CRITERIAGROUP_HANDLE) NULL
         );
40    }
41
42    /* populate the criteria group with
         criteria */
43    /* (upto 5) */
44    printf ( “Enter number of Criteria in
         group ==> “ );
45    numcrit = atoi(gets(input));
46
```

```
47    for ( i = 0 ; i < numcrit ; i++ )
48    {
49       /* create the criteria for a media
         stat field */
50       criteriah = VS_Criteria_Create();
51
52       if ( criteriah ==
         (VST_CRITERIA_HANDLE) NULL )
53       {
54          /* could not allocate handle */
55          rc = VSE_FALSE;
56          break;
57       }
58
59       printf ( "Enter the media's field
         number ==> " );
60       field = atoi(gets(input));
61
62       printf ( "Enter the sort order
         (Ascending - 1, Descending - 2)
         ==> " );
63       sort = atoi(gets(input));
64
65       /* set the criteria parameters */
66       /VS_Criteria_SetFields (
         criteriah,
67                   VSID_FIELD,
         field,
68
         VSID_MOUNT_CRITERIA_ORDER, sort,
69                   VSID_ENDFIELD );
70
71       /* populate the criteria with
         expressions */
72       /* (upto 4) */
73       printf ( "Enter the number of
         criteria expressions ==> " );
74       numexpr = atoi(gets(input));
75
76       for ( j = 0 ; j < numexpr ; j++ )
77       {
```

```
78          /* create an expression for
        this criteria */
79          exprh =
        VS_Expression_Create();
80
81          if ( exprh ==
        (VST_EXPRESSION_HANDLE) NULL )
82          {
83             /* could not allocate memory
        for this */
84             /* handle */
85             rc = VSE_FALSE;
86             break;
87          }
88
89          printf ( “Enter relational
        option (eq 1, gt 2, ge 3, lt 4, le
        5, ne 6) ==> “ );
90          relopt = atoi(gets(input));
91
92          printf ( “Enter the media field
        value ==> “ );
93          gets( value);
94
95          printf ( “Enter connective
        operation (none 0, and 1, or 2)
        ==> “ );
96          conop = atoi(gets(input));
97
98          /* set the expression’s
        parameters */
99          VS_Expression_SetFields (
        exprh,
100                VSID_MOUNT_CRITERIA_OPT,
                        relopt,
101                VSID_CONNECTIVE_OP,
        conop,
102                VSID_MEDIA_STAT_VALUE,
        value,
103                VSID_ENDFIELD );
104
```

```
105         /* add the expression to the
            criteria */
106         VS_Criteria_SetFields (
            criteriah,
107
            VSID_EXPRESSION_HANDLE_ENTRY, j,
            exprh,
108               VSID_ENDFIELD );
109
110      }
111      /* add the criteria to the
            criteria group */
112      VS_CriteriaGroup_SetFields (
            grouph,
113         VSID_CRITERIA_HANDLE_ENTRY, i,
            criteriah,
114         VSID_ENDFIELD );
115   }
116
117   /* if it failed, destroy the criteria
         group handle */
118   if ( rc == VSE_FALSE )
119   {
120      /* criteria group will destroy any
         */
121      /* criteria and their expressions
         */
122      /* for us */
123      VS_CriteriaGroup_Destroy ( grouph
         );
124
125      grouph =
         (VST_CRITERIAGROUP_HANDLE) NULL;
126   }
127
128   return ( grouph );
129}
```

Notes

The `VSID_CRITERIA_HANDLE_ENTRY` parameter requires that two arguments be passed instead of one. The first argument passed is the entry number in the criteria group table. The second argument is the value to be stored.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Criteria_Create(l),
- VS_Criteria_Destroy(l),
- VS_Criteria_GetFields(l),
- VS_Criteria_SetFields(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_GetFields(l),
- VS_Error_GetFields(l),
- VS_Expression_Create(l),
- VS_Expression_Destroy(l),
- VS_Expression_GetFields(l),
- VS_Expression_SetFields(l),
- VSCMD_Mount(l)

VS_Drive_Create

The `VS_Drive_Create` function allocates a VolServ API drive handle. A drive handle is used to pass drive information to and from VolServ.

Synopsis

VST_DRIVE_HANDLE VS_Drive_Create (void)

Arguments

None

Return Values

`VS_Drive_Create` returns:

- A drive handle, if one can be allocated.
- NULL, if a drive handle cannot be allocated. An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_drive_handle
4  *
5  * PURPOSE:
6  * This function tests a drive handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_drive_handle(void)
14 #else
15    VST_BOOLEAN vst_drive_handle()
16 #endif
17 {
```

```
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_DRIVE_HANDLE     h;
20    VST_DRIVE_ID         DriveID;
21    VST_DRIVE_TYPE       DriveType;
22    VST_ARCHIVE_NAME     ArchiveName;
23    VST_COMP_STATE       ComponentState;
24    VST_ASSIGNMENT       Assignment;
25    VST_MOUNT_STATE      MountState;
26    VST_USAGE_COUNT      UsageCount;
27    VST_USAGE            CurrentTime;
28    VST_USAGE            TotalTime;
29    VST_COUNT            ErrorCount;
30    VST_MEDIA_ID         MountedMediaID;
31
32    /* create the handle */
33    h = VS_Drive_Create();
34    if (h != (VST_DRIVE_HANDLE) NULL)
35    {
36       /* get values from user */
37       printf("Enter Drive ID ==> ");
38       DriveID = atoi(gets(input));
39       printf("Enter Drive Type ==> ");
40       DriveType = atoi(gets(input));
41       printf("Enter Associated Archive
         ==> ");
42       gets(ArchiveName);
43       printf("Enter Component State ==>
         ");
44       ComponentState =
         atoi(gets(input));
45       printf("Enter Assignment ==> ");
46       Assignment = atoi(gets(input));
47       printf("Enter Mount State ==> ");
48       MountState = atoi(gets(input));
49       printf("Enter Usage Count ==> ");
50       UsageCount = atoi(gets(input));
51       printf("Enter Current Usage Time
         ==> ");
52       CurrentTime = atoi(gets(input));
53       printf("Enter Total Usage Time ==>
         ");
54       TotalTime = atoi(gets(input));
```

```
55       printf("Enter Error Count ==> ");
56       ErrorCount = atoi(gets(input));
57       printf("Enter Mounted Media ID ==>
         ");
58       gets(MountedMediaID);
59       /* set the fields */
60       rc = VS_Drive_SetFields(h,
61            VSID_DRIVE_ID,
         DriveID,
62            VSID_DRIVE_TYPE,
         DriveType,
63            VSID_ARCHIVE_NAME,
         ArchiveName,
64            VSID_COMP_STATE,
         ComponentState,
65            VSID_ASSIGNMENT,
         Assignment,
66            VSID_MOUNT_STATE,
         MountState,
67            VSID_USAGE_COUNT,
         UsageCount,
68            VSID_USAGE_TIME,
         CurrentTime,
69            VSID_TOTAL_USAGE_TIME,
         TotalTime,
70            VSID_ERROR_COUNT,
         ErrorCount,
71            VSID_MEDIA_ID,
         MountedMediaID,
72            VSID_ENDFIELD);
73       if (rc)
74       {
75          vst_print_drive(h);
76       }
77       VS_Drive_Destroy(h);
78    }
79    return(rc);
80 }
```


Notes None

See Also

- vsapi(l),
- VS_Drive_Destroy(l),
- VS_Drive_GetFields(l),
- VS_Drive_SetFields(l),
- VS_Error_GetFields(l)

VS_Drive_Destroy

The VS_Drive_Destroy deallocates a drive handle that was allocated with VS_Drive_Create. A drive handle is used to pass drive information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Drive_Destroy
(VST_DRIVE_HANDLE handle)

Arguments

- handle = Drive handle to be destroyed.

Return Values

VS_Drive_Destroy returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - The appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a drive handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_drive_handle
4  *
5  * PURPOSE:
6  * This function tests a drive handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
```

```
13    VST_BOOLEAN vst_drive_handle(void)
14 #else
15    VST_BOOLEAN vst_drive_handle()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_DRIVE_HANDLE     h;
20    VST_DRIVE_ID         DriveID;
21    VST_DRIVE_TYPE       DriveType;
22    VST_ARCHIVE_NAME     ArchiveName;
23    VST_COMP_STATE       ComponentState;
24    VST_ASSIGNMENT       Assignment;
25    VST_MOUNT_STATE      MountState;
26    VST_USAGE_COUNT      UsageCount;
27    VST_USAGE            CurrentTime;
28    VST_USAGE            TotalTime;
29    VST_COUNT            ErrorCount;
30    VST_MEDIA_ID         MountedMediaID;
31
32    /* create the handle */
33    h = VS_Drive_Create();
34    if (h != (VST_DRIVE_HANDLE) NULL)
35    {
36       /* get values from user */
37       printf("Enter Drive ID ==> ");
38       DriveID = atoi(gets(input));
39       printf("Enter Drive Type ==> ");
40       DriveType = atoi(gets(input));
41       printf("Enter Associated Archive
         ==> ");
42       gets(ArchiveName);
43       printf("Enter Component State ==>
         ");
44       ComponentState =
         atoi(gets(input));
45       printf("Enter Assignment ==> ");
46       Assignment = atoi(gets(input));
47       printf("Enter Mount State ==> ");
48       MountState = atoi(gets(input));
49       printf("Enter Usage Count ==> ");
50       UsageCount = atoi(gets(input));
```

```
51       printf("Enter Current Usage Time
         ==> ");
52       CurrentTime = atoi(gets(input));
53       printf("Enter Total Usage Time ==>
         ");
54       TotalTime = atoi(gets(input));
55       printf("Enter Error Count ==> ");
56       ErrorCount = atoi(gets(input));
57       printf("Enter Mounted Media ID ==>
         ");
58       gets(MountedMediaID);
59       /* set the fields */
60       rc = VS_Drive_SetFields(h,
61            VSID_DRIVE_ID,
         DriveID,
62            VSID_DRIVE_TYPE,
         DriveType,
63            VSID_ARCHIVE_NAME,
         ArchiveName,
64            VSID_COMP_STATE,
         ComponentState,
65            VSID_ASSIGNMENT,
         Assignment,
66            VSID_MOUNT_STATE,
         MountState,
67            VSID_USAGE_COUNT,
         UsageCount,
68            VSID_USAGE_TIME,
         CurrentTime,
69            VSID_TOTAL_USAGE_TIME,
         TotalTime,
70            VSID_ERROR_COUNT,
         ErrorCount,
71            VSID_MEDIA_ID,
         MountedMediaID,
72            VSID_ENDFIELD);
73       if (rc)
74       {
75          vst_print_drive(h);
76       }
77       VS_Drive_Destroy(h);
78    }
```

```
79    return(rc);
80 }
```

Notes

After `VS_Drive_Destroy` has been called for a drive handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Drive_Create(l),
- VS_Drive_GetFields(l),
- VS_Drive_SetFields(l),
- VS_Error_GetFields(l)

VS_Drive_Get Fields

`VS_Drive_GetFields` retrieves information associated with a drive handle. A drive handle is used to pass drive information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Drive_GetFields
(VST_DRIVE_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Drive handle for which information is being requested.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Pointer to the name of the archive with which this drive is associated. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ASSIGNMENT (VST_ASSIGNMENT *)	Pointer to the current assignment of this drive. Valid `VSID_ASSIGNMENT` values are enumerated in the *`vs_types.h`* file.

Parameter Type	Description
VSID_COMP_STATE (VST_COMP_STATE *)	Pointer to the operational state of this drive. Valid VSID_COMP_STATE values are enumerated in the *vs_types.h* file.
VSID_DRIVE_ID (VST_DRIVE_ID *)	Pointer to the identifier of this drive.
VSID_DRIVE_TYPE (VST_DRIVE_TYPE *)	Pointer to the type of this drive. Valid VSID_DRIVE_TYPE values are enumerated in the *vs_types.h* file.
VSID_ERROR_COUNT (VST_COUNT *)	Pointer to the error count of the drive.
VSID_MEDIA_ID (VST_MEDIA_ID)	If the VSID_MOUNT_STATE of this drive is VSE_MOUNT_MOUNTED, identifier of the medium mounted on this drive.
VSID_MEDIA_TYPE_ENTRY (int)	Index of a specific media type handle in the media type handle table.
(VST_MEDIATYPE_HANDLE *)	Pointer to the location where the media type handle should be stored.
VSID_MEDIA_TYPE_TABLE (VST_TABLE_HANDLE *)	Pointer to the media types (in table format) supported by this drive.
VSID_MOUNT_STATE (VST_MOUNT_STATE *)	Pointer to the mount state of this drive. Valid VSID_MOUNT_STATE values are enumerated in the *vs_types.h* file.
VSID_NUMBER_MEDIA TYPES (int *)	Pointer to the number of media types present in the media type name table.
VSID_USAGE_COUNT (VST_USAGE_COUNT *)	Pointer to the number of times this drive has been mounted.
VSID_USAGE_TIME (VST_USAGE *)	Pointer to the current usage time of the drive.
VSID_TOTAL_USAGE_TIME (VST_USAGE *)	Pointer to the total usage time of the drive.

Return Values VS_Drive_GetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - The appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a drive handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
- VSE_ERR_OUTOFRANGE - An index value was out of range.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_drive
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a drive handle.
8  *
9  * PARAMETERS:
10 * h : the drive handle to print
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    void
         vst_print_drive(VST_DRIVE_HANDLE
         h)
15 #else
16    void vst_print_drive(h)
17    VST_DRIVE_HANDLE h;
18 #endif
19 {
```

```
20    VST_DRIVE_ID         DriveID;
21    VST_DRIVE_TYPE       DriveType;
22    VST_ARCHIVE_NAME     ArchiveName;
23    VST_COMP_STATE       ComponentState;
24    VST_ASSIGNMENT       Assignment;
25    VST_MOUNT_STATE      MountState;
26    VST_USAGE_COUNT      UsageCount;
27    VST_USAGE            CurrentTime;
28    VST_USAGE            TotalTime;
29    VST_COUNT            ErrorCount;
30    VST_MEDIA_ID         MountedMediaID;
31    char                 * MediaType;
32    VST_TABLE_HANDLE     MediaTypeTable;
33    int                  i;
34    int                  n;
35
36    VS_Drive_GetFields(h,
37          VSID_DRIVE_ID,
         &DriveID,
38          VSID_DRIVE_TYPE,
         &DriveType,
39          VSID_ARCHIVE_NAME,
         ArchiveName,
40          VSID_COMP_STATE,
         &ComponentState,
41          VSID_ASSIGNMENT,
         &Assignment,
42          VSID_MOUNT_STATE,
         &MountState,
43          VSID_USAGE_COUNT,
         &UsageCount,
44          VSID_USAGE_TIME,
         &CurrentTime,
45          VSID_TOTAL_USAGE_TIME,
         &TotalTime,
46          VSID_ERROR_COUNT,
         &ErrorCount,
47          VSID_MEDIA_ID,
         MountedMediaID,
48          VSID_MEDIA_TYPE_TABLE,
         &MediaTypeTable,
49          VSID_ENDFIELD);
```

```
50    printf("******Drive
         Handle******\n");
51    printf("Drive ID = %dDriveType =
         %d\n",DriveID,DriveType);
52    printf("ArchiveName =
         %s\n",ArchiveName);
53    printf("Comp State = %dAssignment =
         %d\n",ComponentState,Assignment);
54    printf("Mount State = %dUsageCount =
         %d\n",MountState,UsageCount);
55    printf("Current Usage = %dTotal
         Usage = %d\n", CurrentTime,
         TotalTime);
56    printf("Error Count = %d\n",
         ErrorCount );
57    printf("MediaID =
         %s\n",MountedMediaID);
58
59     /* DrivePoolQuery Doesn't use this
         Field */
60    if (MediaTypeTable !=
         (VST_TABLE_HANDLE)NULL)
61    {
62
         VS_Table_GetFields(MediaTypeTable
         ,
63
         VSID_NUMBER_ENTRIES, &n,
64                          VSID_ENDFIELD);
65       for ( i = 0; i < n; i++)
66       {
67
         VS_Table_GetFields(MediaTypeTable
         ,
68                   VSID_TABLE_ENTRY, i,
         &MediaType,
69                   VSID_ENDFIELD);
70          printf("MediaType Entry #%d =
         %s\n",i,MediaType);
71       }
72    }
73 }
```

Notes

VolServ may place a drive in the `VSE_COMP_UNAVAIL` state when a parent component goes off-line.

The `VSID_MEDIA_TYPE_ENTRY` parameter requires that two arguments be passed instead of one. The first argument passed is the entry number in the drive table. The second argument is a pointer to the location where the value is stored.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Drive_Create(l),
- VS_Drive_Destroy(l),
- VS_Drive_SetFields(l),
- VS_Error_GetFields(l),
- VS_MediaType_GetFields(l),
- VS_MediaType_SetFields(l),
- VS_Table_GetFields(l),
- VSCMD_DriveQuery(l),
- VSCMD_DrivePoolQuery(l)

VS_Drive_Set Fields

`VS_Drive_SetFields` sets the value of one or more fields in a drive handle. A drive handle is used to pass information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Drive_SetFields
(VST_DRIVE_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Drive handle where information is stored.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive with which this drive is associated. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ASSIGNMENT (VST_ASSIGNMENT)	Current assignment of this drive. Valid `VSID_ASSIGNMENT` values are enumerated in the *`vs_type.h`* file.

Parameter Type	Description
VSID_COMP_STATE (VST_COMP_STATE)	Operational state of this archive. Valid VSID_COMP_STATE values are enumerated in the *vs_types.h* file.
VSID_DRIVE_ID (VST_DRIVE_ID)	Identifier of this drive.
VSID_DRIVE_TYPE (VST_DRIVE_TYPE)	Type of this drive. Valid VSID_DRIVE_TYPE values are enumerated in the *vs_types.h* file.
VSID_ERROR_COUNT (VST_COUNT)	Error count of the drive.
VSID_MEDIA_ID (VST_MEDIA_ID)	If the VSID_MOUNT_STATE of this drive is VSE_MOUNT_MOUNTED, the identifier of the medium mounted on this drive.
VSID_MEDIA_TYPE_TABLE (VST_TABLE_HANDLE)	Media types (in table format) supported by this drive.
VSID_MOUNT_STATE (VST_MOUNT_STATE)	Mount state of this drive. Valid VSID_MOUNT_STATE values are enumerated in the *vs_types.h* file.
VSID_USAGE_COUNT (VST_USAGE_COUNT)	Number of times this drive has been mounted.
VSID_USAGE_TIME (VST_USAGE_TIME)	Current usage time of the drive.
VSID_TOTAL_USAGE_TIME (VST_USAGE)	Total usage time of the drive.

Return Values

VS_Drive_SetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - The appropriate error code is set in VSG_Error
- VSE_ERR_BADFIELD - An invalid parameter was specified.

- VSE_ERR_BADHANDLE - Specified handle was not a drive handle.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_drive_handle
4  *
5  * PURPOSE:
6  * This function tests a drive handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_drive_handle(void)
14 #else
15    VST_BOOLEAN vst_drive_handle()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_DRIVE_HANDLE     h;
20    VST_DRIVE_ID         DriveID;
21    VST_DRIVE_TYPE       DriveType;
22    VST_ARCHIVE_NAME     ArchiveName;
23    VST_COMP_STATE       ComponentState;
24    VST_ASSIGNMENT       Assignment;
25    VST_MOUNT_STATE      MountState;
26    VST_USAGE_COUNT      UsageCount;
```

```
27    VST_USAGE            CurrentTime;
28    VST_USAGE            TotalTime;
29    VST_COUNT            ErrorCount;
30    VST_MEDIA_ID         MountedMediaID;
31
32    /* create the handle */
33    h = VS_Drive_Create();
34    if (h != (VST_DRIVE_HANDLE) NULL)
35    {
36       /* get values from user */
37       printf("Enter Drive ID ==> ");
38       DriveID = atoi(gets(input));
39       printf("Enter Drive Type ==> ");
40       DriveType = atoi(gets(input));
41       printf("Enter Associated Archive
         ==> ");
42       gets(ArchiveName);
43       printf("Enter Component State ==>
         ");
44       ComponentState =
         atoi(gets(input));
45       printf("Enter Assignment ==> ");
46       Assignment = atoi(gets(input));
47       printf("Enter Mount State ==> ");
48       MountState = atoi(gets(input));
49       printf("Enter Usage Count ==> ");
50       UsageCount = atoi(gets(input));
51       printf("Enter Current Usage Time
         ==> ");
52       CurrentTime = atoi(gets(input));
53       printf("Enter Total Usage Time ==>
         ");
54       TotalTime = atoi(gets(input));
55       printf("Enter Error Count ==> ");
56       ErrorCount = atoi(gets(input));
57       printf("Enter Mounted Media ID ==>
         ");
58       gets(MountedMediaID);
59       /* set the fields */
60       rc = VS_Drive_SetFields(h,
61             VSID_DRIVE_ID,
         DriveID,
```

```
62              VSID_DRIVE_TYPE,
          DriveType,
63              VSID_ARCHIVE_NAME,
          ArchiveName,
64              VSID_COMP_STATE,
          ComponentState,
65              VSID_ASSIGNMENT,
          Assignment,
66              VSID_MOUNT_STATE,
          MountState,
67              VSID_USAGE_COUNT,
          UsageCount,
68              VSID_USAGE_TIME,
          CurrentTime,
69              VSID_TOTAL_USAGE_TIME,
          TotalTime,
70              VSID_ERROR_COUNT,
          ErrorCount,
71              VSID_MEDIA_ID,
          MountedMediaID,
72              VSID_ENDFIELD);
73        if (rc)
74        {
75           vst_print_drive(h);
76        }
77        VS_Drive_Destroy(h);
78     }
79     return(rc);
80 }
```

Notes

VolServ may place a drive in the `VSE_COMP_UNAVAIL` state when a parent component goes off-line. The `VSE_COMP_UNAVAIL` state cannot be specified by the user.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Drive_Create(l),
- VS_Drive_Destroy(l),
- VS_Drive_GetFields(l),
- VS_Error_GetFields(l),
- VS_MediaType_GetFields(l),
- VS_MediaType_SetFields(l),
- VS_Table_GetFields(l)

VS_DrivePool_Create

`VS_DrivePool_Create` allocates a VolServ API drive pool handle. A drive pool handle is used to pass drive pool information to and from VolServ.

Synopsis

VST_DRIVEPOOL_HANDLE VS_DrivePool_Create
(void)

Arguments

None

Return Values

`VS_DrivePool_Create` returns:

- A drive pool handle, if one can be allocated.
- NULL, if a drive pool handle cannot be allocated. An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_drivepool_handle
4  *
5  * PURPOSE:
6  * This function tests a drive pool
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_drivepool_handle(void)
14 #else
```

```
15    VST_BOOLEAN
         vst_drivepool_handle(void)
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_DRIVEPOOL_HANDLE    h;
20    VST_DRIVE_POOL_NAME
         DrivePoolName;
21
22    /* create the handle */
23    h = VS_DrivePool_Create();
24    if (h != (VST_DRIVEPOOL_HANDLE) NULL)
25    {
26       /* get values from user */
27       printf("*** Drive Pool Handle
         ***\n");
28       printf("Enter Drive Pool Name ==>
         ");
29       gets(DrivePoolName);
30       rc = VS_DrivePool_SetFields(h,
31                VSID_DRIVEPOOL_NAME,
         DrivePoolName,
32                VSID_ENDFIELD);
33       if (rc)
34       {
35          vst_print_drivepool(h);
36       }
37       VS_DrivePool_Destroy(h);
38    }
39    return(rc);
40 }
```

Notes None

See Also

- vsapi(l),
- VS_DrivePool_Destroy(l),
- VS_DrivePool_GetFields(l),
- VS_DrivePool_SetFields(l),
- VS_Error_GetFields(l)

VS_DrivePool_Destroy

`VS_DrivePool_Destroy` deallocates a drive pool handle that was allocated with `VS_DrivePool_Create`. A drive pool handle is used to pass drive pool information to and from VolServ.

Synopsis

VST_BOOLEAN VS_DrivePool_Destroy
(VST_DRIVEPOOL_HANDLE handle)

Arguments

- `handle` = Drive pool handle to be destroyed.

Return Values

`VS_DrivePool_Destroy` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - The appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADHANDLE` - Specified handle was not a drive pool handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_drivepool_handle
4  *
5  * PURPOSE:
6  * This function tests a drive pool
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_drivepool_handle(void)
14 #else
15    VST_BOOLEAN
         vst_drivepool_handle(void)
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_DRIVEPOOL_HANDLE    h;
20    VST_DRIVE_POOL_NAME
         DrivePoolName;
21
22    /* create the handle */
23    h = VS_DrivePool_Create();
24    if (h != (VST_DRIVEPOOL_HANDLE) NULL)
25    {
26       /* get values from user */
27       printf("*** Drive Pool Handle
         ***\n");
28       printf("Enter Drive Pool Name ==>
         ");
29       gets(DrivePoolName);
30       rc = VS_DrivePool_SetFields(h,
31                VSID_DRIVEPOOL_NAME,
         DrivePoolName,
32                VSID_ENDFIELD);
33       if (rc)
34       {
35          vst_print_drivepool(h);
36       }
37       VS_DrivePool_Destroy(h);
38    }
39    return(rc);
40 }
```

Notes

After `VS_DrivePool_Destroy` has been called for a drive pool handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_DrivePool_Create(l),
- VS_DrivePool_GetFields(l),
- VS_DrivePool_SetFields(l),
- VS_Error_GetFields(l)

VS_DrivePool _GetFields

VS_DrivePool_GetFields retrieves information associated with a drive pool handle. A drive pool handle is used to pass drive pool information to and from VolServ.

Synopsis

VST_BOOLEAN VS_DrivePool_GetFields
(VST_DRIVEPOOL_HANDLE handle,
“…”,
VSID_ENDFIELD)

Arguments

- handle = Drive pool handle for which information is retrieved.
- “…” = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_DRIVE_HANDLE_ENTRY (int)	Index of the drive handle to retrieve.
(VST_DRIVE_HANDLE *)	Pointer to the location to store the drive handle.
VSID_DRIVE_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the drives (in table format) that belong to this drive pool group.
VSID_DRIVE_ID (VST_DRIVE_ID *)	Pointer to the first drive id in the drive handle table.

Parameter Type	Description
VSID_DRIVE_ID_Entry (int, VST_DRIVE_ID *)	Index of the drive in the drive handle table. Pointer to the location to store the drive identifier.
VSID_DRIVEPOOL_NAME (VST_DRIVEPOOL_NAME)	Pointer to the name associated with the drive pool group. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_NUMBER_DRIVE_HANDLES (int *)	Pointer to the number of drive handles in the drive handle table.

Return Values

`VS_DrivePool_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - The appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a drive pool handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_OUTOFRANGE` - An index value was out of range.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_drivepool
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * in a drive pool handle.
8  *
9  * PARAMETERS:
10 * h : the drive pool handle to print
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    void
         vst_print_drivepool(VST_DRIVEPOOL
         _HANDLE h)
15 #else
16    void vst_print_drivepool(h)
17    VST_DRIVEPOOL_HANDLE h;
18 #endif
19 {
20    VST_DRIVE_POOL_NAME  DrivePoolName;
21    VST_TABLE_HANDLE
         DriveHandleTable;
22    VST_DRIVE_HANDLE     DriveHandle;
23    int                  i;
24    int                  n;
25
26    VS_DrivePool_GetFields(h,
27          VSID_DRIVEPOOL_NAME,
         DrivePoolName,
28          VSID_DRIVE_HANDLE_TABLE
         &DriveHandleTable,
29          VSID_ENDFIELD);
30    printf("DrivePoolName =
         %s\n",DrivePoolName);
31    /* Get # of entries */
32    if ( DriveHandleTable !=
         (VST_TABLE_HANDLE) NULL )
```

```
33    {
34
          VS_Table_GetFields(DriveHandleTab
          le,
35           VSID_NUMBER_ENTRIES,    &n,
36           VSID_ENDFIELD);
37        for ( i = 0; i < n; i++)
38        {
39
          VS_Table_GetFields(DriveHandleTab
          le,
40              VSID_TABLE_ENTRY, i,
          &DriveHandle,
41              VSID_ENDFIELD);
42           vst_print_drive(DriveHandle);
43        }
44    }
45 }
```

Notes

The `VSID_DRIVE_HANDLE_ENTRY` parameter requires that two arguments be passed instead of one. The first argument passed is the entry number in the drive pool table. The second argument is a pointer to the location where the value is stored.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_DrivePool_Create(l),
- VS_DrivePool_Destroy(l),
- VS_DrivePool_SetFields(l),
- VS_Error_GetFields(l),
- VS_Table_GetFields(l)

- VSCMD_DrivePoolQuery(l)

VS_DrivePool_SetFields

`VS_DrivePool_SetFields` sets the value of one or more fields in a drive pool handle. A drive pool handle is used to pass drive pool information to and from VolServ.

Synopsis

VST_BOOLEAN VS_DrivePool_SetFields
(VST_DRIVEPOOL_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Drive pool handle where information is stored.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_DRIVE_HANDLE_TABLE (VST_TABLE_HANDLE)	Drive handles (in table format) that belong to this drive pool group.
VSID_DRIVEPOOL_NAME (VST_DRIVEPOOL_NAME)	Name associated with the drive pool group. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Return Values

`VS_DrivePool_SetFields` returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - The appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a drive pool handle.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_drivepool_handle
4  *
5  * PURPOSE:
6  * This function tests a drive pool
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_drivepool_handle(void)
14 #else
15    VST_BOOLEAN
         vst_drivepool_handle(void)
```

```
#endif
{
   VST_BOOLEAN             rc =
      VSE_FALSE;
   VST_DRIVEPOOL_HANDLE    h;
   VST_DRIVE_POOL_NAME
      DrivePoolName;

   /* create the handle */
   h = VS_DrivePool_Create();
   if (h != (VST_DRIVEPOOL_HANDLE) NULL)
   {
      /* get values from user */
      printf("*** Drive Pool Handle
      ***\n");
      printf("Enter Drive Pool Name ==>
      ");
      gets(DrivePoolName);
      rc = VS_DrivePool_SetFields(h,
               VSID_DRIVEPOOL_NAME,
      DrivePoolName,
               VSID_ENDFIELD);
      if (rc)
      {
         vst_print_drivepool(h);
      }
      VS_DrivePool_Destroy(h);
   }
   return(rc);
}
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_DrivePool_Create(l),
- VS_DrivePool_Destroy(l),
- VS_DrivePool_GetFields(l),
- VS_Error_GetFields(l),
- VS_Table_Getfields(l)

VS_Error_Get Fields

`VS_Error_GetFields` retrieves information associated with an error handle. An error handle is used to pass error information to and from VolServ.

An error handle is associated with each command handle and with each notify handle. There is also a global error handle for errors that cannot be associated with a command.

After a VolServ request or an attempt to receive callbacks fails, information is stored in an error handle. `VS_Error_GetFields` allows the user to retrieve this information.

Synopsis

```
VST_BOOLEAN VS_Error_GetFields
( VST_ERROR_HANDLE handle,
"…",
VSID_ENDFIELD )
```

Arguments

- `handle` = The error handle where information is retrieved.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = `Re`quired at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ERROR_CODE (VST_ERROR_CODE)	Pointer to the error code for the given error.
VSID_ERROR_FILE (VST_ERROR_FILE)	Name of the source file where the error occurred (API internal errors only).
VSID_ERROR_LINE (int *)	Pointer to the source line number where the error occurred (API internal errors only).
VSID_ERROR_NUMBER (VST_ERROR_NUMCODE *)	Pointer to the field that indicates which error occurred.
VSID_ERROR_OBJECT (VST_ERROR_OBJCODE *)	Pointer to the field that indicates the location of the error.

Return Values

`VS_Error_GetFields` returns

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - The appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not an error handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_error
4  *
5  * PURPOSE:
```

```
6  * This function prints out the
       information stored in
7  * an error handle.
8  *
9  * PARAMETERS:
10 * h : the error handle to print
11 *
12 ************************************
       **********/
13 #ifdef ANSI_C
14    void
         vst_print_error(VST_ERROR_HANDLE
         h)
15 #else
16    void vst_print_error(h)
17    VST_ERROR_HANDLE h;
18 #endif
19 {
20    VST_ERROR_CODE    err;
21    int               line;
22    VST_ERROR_FILE    file;
23
24    VS_Error_GetFields(h,
25                       VSID_ERROR_CODE,
         err,
26                       VSID_ERROR_LINE,
         &line,
27                       VSID_ERROR_FILE,
         file,
28                       VSID_ENDFIELD);
29
30    printf("******Error
         Handle******\n");
31    printf("Error Code = %s\n",err);
32    printf("Error File = %s\n",file);
33    printf("Error Line = %d\n",line);
34 }
```

Notes

The VSID_ERROR_OBJECT parameter tells specifically where (in VolServ or an API object) the error occurred. If its value is VSE_VOLSERV, the VSID_ERROR_NUMBER parameter matches a VolServ error code in the VST_VOLERR_CODE error code. If not, the VSID_ERROR_NUMBER parameter matches a value in the VST_ERROR_NUMCODE type.

The VSID_ERROR_FILE and VSID_ERROR_LINE parameters are valid only for API internal errors (e.g., VSE_ERR_OUTOFMEM).

The VSID_ERROR_CODE parameter is a human-readable code that tells where the error occurred and gives the error number. It is in the form AAAnnn, where AAA is an abbreviation that corresponds with the VST_ERROR_OBJCODE type. If this abbreviation is VOL, the error number is a VolServ error code. Otherwise, it is in the VST_ERROR_NUMCODE type.

If the string parameters are not sufficiently long, unpredictable results occur.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_GetFields(l),
- VS_Notify_GetFields(l)

VS_ Expression_ Create

`VS_Expression_Create` allocates a VolServ API expression handle. An expression handle is used to pass expression information to and from VolServ.

An expression handle has three parts: a relation operator (=, >, <, >=, <=, <>), a connective option (and/or/none), and a comparison value. A criteria handle uses expression handles to build the comparison function when using mount-by-selection criteria.

Synopsis

VST_EXPRESSION_HANDLE VS_Expression_Create
(void)

Arguments

- `handle` = The expression handle to be created.

Return Values

`VS_Expression_Create` returns:

- An expression handle, if one can be allocated.
- NULL, if the expression handle cannot be allocated. An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_criteria
4  *
5  * PURPOSE:
6  * This function creates the mount
       criteria group
7  * handle and sets the values in it
       according to
8  * user input.
```

```
9  *
10 * PARAMETERS:
11 * none
12 *
13 ************************************
       *********/
14 #ifdef ANSI_C
15    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria(void)
16 #else
17    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria()
18 #endif
19 {
20    int                        i;
21    int                        j;
22    int                        numcrit;
23    int                        numexpr;
24    VST_BOOLEAN                rc =
         VSE_TRUE;
25    VST_EXPRESSION_HANDLE      exprh;
26    VST_CRITERIA_HANDLE
         criteriah;
27    VST_CRITERIAGROUP_HANDLE   grouph;
28    VST_COUNT                  field;
29    VST_MOUNT_CRITERIA_ORDER   sort;
30    VST_MEDIA_STAT_VALUE       value;
31    VST_MOUNT_CRITERIA_OPT     relopt;
32    VST_CONNECTIVE_OP          conop;
33
34    /* create the criteria group */
35    grouph = VS_CriteriaGroup_Create();
36
37    if ( grouph ==
         (VST_CRITERIAGROUP_HANDLE) NULL )
38    {
39       /* out of memory -- return */
40       return (
         (VST_CRITERIAGROUP_HANDLE) NULL
         );
41    }
42
```

```
43    /* populate the criteria group with
         criteria */
44    /* (upto 5) */
45    printf ( “Enter number of Criteria in
         group ==> “ );
46    numcrit = atoi(gets(input));
47
48    for ( i = 0 ; i < numcrit ; i++ )
49    {
50       /* create the criteria for a media
         stat field */
51       criteriah = VS_Criteria_Create();
52
53       if ( criteriah ==
         (VST_CRITERIA_HANDLE) NULL )
54       {
55          /* could not allocate handle */
56          rc = VSE_FALSE;
57          break;
58       }
59
60       printf ( “Enter the media’s field
         number ==> “ );
61       field = atoi(gets(input));
62
63       printf ( “Enter the sort order
         (Ascending - 1, Descending - 2)
         ==> “ );
64       sort = atoi(gets(input));
65
66       /* set the criteria parameters */
67       /VS_Criteria_SetFields (
         criteriah,
68                   VSID_FIELD,
         field,
69
         VSID_MOUNT_CRITERIA_ORDER, sort,
70                   VSID_ENDFIELD );
71
72       /* populate the critera with
         expressions */
73       /* (upto 4) */
```

```
74       printf ( “Enter the number of
         criteria expressions ==> “ );
75       numexpr = atoi(gets(input));
76
77       for ( j = 0 ; j < numexpr ; j++ )
78       {
79          /* create an expression for
         this criteria */
80          exprh =
         VS_Expression_Create();
81
82          if ( exprh ==
         (VST_EXPRESSION_HANDLE) NULL )
83          {
84             /* could not allocate memory
         for this */
85             /* handle */
86             rc = VSE_FALSE;
87             break;
88          }
89
90          printf ( “Enter relational
         option (eq 1, gt 2, ge 3, lt 4, le
         5, ne 6) ==> “ );
91          relopt = atoi(gets(input));
92
93          printf ( “Enter the media field
         value ==> “ );
94          gets( value);
95
96          printf ( “Enter connective
         operation (none 0, and 1, or 2)
         ==> “ );
97          conop = atoi(gets(input));
98
99          /* set the expression’s
         parameters */
100         VS_Expression_SetFields (
         exprh,
101            VSID_MOUNT_CRITERIA_OPT,
         relopt,
```

```
102           VSID_CONNECTIVE_OP,
        conop,
103           VSID_MEDIA_STAT_VALUE,
        value,
104           VSID_ENDFIELD );
105
106        /* add the expression to the
        criteria */
107        VS_Criteria_SetFields (
        criteriah,
108
        VSID_EXPRESSION_HANDLE_ENTRY, j,
        exprh,
109           VSID_ENDFIELD );
110
111     }
112     /* add the criteria to the
        criteria group */
113     VS_CriteriaGroup_SetFields (
        grouph,
114        VSID_CRITERIA_HANDLE_ENTRY, i,
        criteriah,
115        VSID_ENDFIELD );
116  }
117
118  /* if it failed, destroy the criteria
        group handle */
119  if ( rc == VSE_FALSE )
120  {
121     /* criteria group will destroy any
        */
122     /* criteria and their expressions
        */
123     /* for us */
124     VS_CriteriaGroup_Destroy ( grouph
        );
125
126     grouph =
        (VST_CRITERIAGROUP_HANDLE) NULL;
127  }
128
129  return ( grouph );
```

```
130}
```

Notes

None

See Also

- vsapi(l),
- VS_Criteria_Create(l),
- VS_Criteria_Destroy(l),
- VS_Criteria_GetFields(l),
- VS_Criteria_SetFields(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_GetFields(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Expression_Destroy(l),
- VS_Expression_GetFields(l),
- VS_Expression_SetFields(l),
- VSCMD_Mount(l)

VS_ Expression_ Destroy

`VS_Expression_Destroy` deallocates an expression handle that was allocated with `VS_Expression_Create`. An expression handle is used to pass expression information to and from VolServ.

Synopsis

VST_BOOLEAN VS_EXPRESSION_DESTROY (VST_EXPRESSION_HANDLE handle)

Arguments

- `handle` = The expression handle to be destroyed.

Return Values

`VS_Expression_Destroy` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - The appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADHANDLE` - Specified handle was not an expression handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_criteria
4  *
5  * PURPOSE:
6  * This function creates the mount
       criteria group handle
7  * and sets the values in it according to
       user input.
8  *
9  * PARAMETERS:
```

```
10 * none
11 *
12 ****************************************
       *********/
13 #ifdef ANSI_C
14    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria(void)
15 #else
16    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria()
17 #endif
18 {
19    int                        i;
20    int                        j;
21    int                        numcrit;
22    int                        numexpr;
23    VST_BOOLEAN                rc =
         VSE_TRUE;
24    VST_EXPRESSION_HANDLE      exprh;
25    VST_CRITERIA_HANDLE
         criteriah;
26    VST_CRITERIAGROUP_HANDLE   grouph;
27    VST_COUNT                  field;
28    VST_MOUNT_CRITERIA_ORDER   sort;
29    VST_MEDIA_STAT_VALUE       value;
30    VST_MOUNT_CRITERIA_OPT     relopt;
31    VST_CONNECTIVE_OP          conop;
32
33    /* create the criteria group */
34    grouph = VS_CriteriaGroup_Create();
35    if ( grouph ==
         (VST_CRITERIAGROUP_HANDLE) NULL )
36    {
37       /* out of memory -- return */
38       return (
         (VST_CRITERIAGROUP_HANDLE) NULL
         );
39    }
40    /* populate the criteria group with
         criteria */
41    /* (upto 5) */
```

```
42    printf ( “Enter number of Criteria in
         group ==> “ );
43    numcrit = atoi(gets(input));
44    for ( i = 0 ; i < numcrit ; i++ )
45    {
46       /* create the criteria for a media
         stat field */
47       criteriah = VS_Criteria_Create();
48       if ( criteriah ==
         (VST_CRITERIA_HANDLE) NULL )
49       {
50          /* could not allocate handle */
51          rc = VSE_FALSE;
52          break;
53       }
54       printf ( “Enter the media’s field
         number ==> “ );
55       field = atoi(gets(input));
56       printf ( “Enter the sort order
         (Ascending - 1, Descending - 2)
         ==>“ );
57       sort = atoi(gets(input));
58       /* set the criteria parameters */
59       VS_Criteria_SetFields (
         criteriah,
60                VSID_FIELD,
         field,
61
         VSID_MOUNT_CRITERIA_ORDER, sort,
62                VSID_ENDFIELD );
63       /* populate the critera with
         expressions */
64       /* (upto 4) */
65       printf ( “Enter the number of
         criteria expressions ==> “ );
66       numexpr = atoi(gets(input));
67       for ( j = 0 ; j < numexpr ; j++ )
68       {
69          /* create an expression for
         this criteria */
70          exprh =
         VS_Expression_Create();
```

```
71          if ( exprh ==
        (VST_EXPRESSION_HANDLE) NULL )
72          {
73             /* could not allocate memory
        for this */
74             /* handle */
75             rc = VSE_FALSE;
76             break;
77          }
78          printf ( “Enter relational
        option (eq 1, gt 2, ge 3, lt 4, le
        5, ne 6) ==> “ );
79          relopt = atoi(gets(input));
80          printf ( “Enter the media field
        value ==> “ );
81          gets( value);
82          printf ( “Enter connective
        operation (none 0, and 1, or 2)
        ==> “ );
83          conop = atoi(gets(input));
84          /* set the expression’s
        parameters */
85          VS_Expression_SetFields (
        exprh,
86                  VSID_MOUNT_CRITERIA_OPT,
                        relopt,
87                  VSID_CONNECTIVE_OP,
        conop,
88                  VSID_MEDIA_STAT_VALUE,
        value,
89                  VSID_ENDFIELD );
90          /* add the expression to the
        criteria */
91          VS_Criteria_SetFields (
        criteriah,
92
        VSID_EXPRESSION_HANDLE_ENTRY, j,
        exprh,
93             VSID_ENDFIELD );
94       }
95       /* add the criteria to the
        criteria group */
```

```
96       VS_CriteriaGroup_SetFields (
         grouph,
97          VSID_CRITERIA_HANDLE_ENTRY, i,
         criteriah,
98          VSID_ENDFIELD );
99    }
100   /* if it failed, destroy the criteria
         group handle */
101   if ( rc == VSE_FALSE )
102   {
103      /* criteria group will destroy any
         */
104      /* criteria and their expressions
         */
105      /* for us, so the only thing that
         is really */
106      /* needed here is a call to */
107      /* VS_CriteriaGroup_Destroy. */
108      /* This is written out the 'long
         way' for */
109      /* documentation purposes. First,
         get the */
110      /* number of criteria */
111
         VS_CriteriaGroup_GetFields(grouph
         ,
112            VSID_NUMBER_ENTRIES,
         &numcrit,
113            VSID_ENDFIELD);
114      for (i = 0; i < numcrit; i++)
115      {
116         /* get a criteria handle */
117
         VS_CriteriaGroup_GetFields(grouph
         ,
118            VSID_CRITERIA_HANDLE_ENTRY,
         i, &criteriah,
119            VSID_ENDFIELD);
120         /* get the number of
         expressions */
121
         VS_Criteria_GetFields(criteriah,
```

```
122           VSID_NUMBER_ENTRIES,
        &numexpr,
123           VSID_ENDFIELD);
124        for (j = 0; j < numexpr; j++)
125        {
126           /* get the expressions from
        the criteria */
127
        VS_Criteria_GetFields(criteriah,
128
        VSID_EXPRESSION_HANDLE_ENTRY, j,
        &exprh,
129           VSID_ENDFIELD);
130           /* destroy the expression
        handle */
131
        VS_Expression_Destroy(exprh);
132           /* let Criteria handle know
        that the */
133           /* expression handle has
        been destroyed */
134
        VS_Criteria_SetFields(criteriah,
135
        VSID_EXPRESSION_HANDLE_ENTRY, j,
        NULL,
136           VSID_ENDFIELD);
137        }
138        /* now, destroy Criteria
        handle. */
139
        VS_Criteria_Destroy(criteriah);
140        /*let the criteria group handle
        know */
141        /* that Criteria handle has
        been */
142        /* destroyed. */
143
        VS_CriteriaGroup_SetFields(grouph
        ,
144           VSID_CRITERIA_HANDLE_ENTRY,
        i, NULL,
```

```
145             VSID_ENDFIELD);
146       }
147       /* finally, destroy the criteria
          group handle. */
148       VS_CriteriaGroup_Destroy ( grouph
          );
149       grouph =
          (VST_CRITERIAGROUP_HANDLE) NULL;
150    }
151    return ( grouph );
152}
```

Notes

After `VS_Expression_Destroy` has been called for an expression handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Criteria_Create(l),
- VS_Criteria_Destroy(l),
- VS_Criteria_GetFields(l)
- VS_Criteria_SetFields(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_GetFields(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Expression_Create(l),
- VS_Expression_GetFields(l),
- VS_Expression_SetFields(l),
- VSCMD_Mount(l)

VS_ Expression_ GetFields

`VS_Expression_GetFields` retrieves information associated with an expression handle. An expression handle is used to pass expression information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Expression_GetFields
(VST_EXPRESSION_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The expression handle where information is retrieved.
- *"..."* = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_MOUNT_CRITERIA_OP (VST_MOUNT_CRITERIA_OPT *)	Pointer to the relational operation for this expression:

Parameter Type	Description
VSID_CONNECTIVE_OP (VST_CONNECTIVE_OP *)	Pointer to the connective operation between two expressions. Valid `VS_Expression_GetFields` values are enumerated in the *`vs_types.h`* file. The last expression in a criteria must have the connect operator set to none.
VSID_MEDIA_STAT_VALUE (VST_MEDIA_STAT_VALUE)	Value for comparison. All values are compared as strings. If numbers are needed, they must be left filled with zeros within the string.

Return Values

`VS_Expression_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - The appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not an error handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_criteria_group
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a criteria group handle.
8  *
```

```
9  * PARAMETERS:
10 * grouph : the mount handle to print
11 *
12 ************************************
       *********/
13 #ifdef ANSI_C
14    void
         vst_print_criteria_group(VST_CRIT
         ERIAGROUP_HANDLE grouph)
15 #else
16    void
         vst_print_criteria_group(grouph)
17    VST_CRITERIAGROUP_HANDLE grouph;
18 #endif
19 {
20    int                          i, j;
21    int                          numcrit =
         0;
22    int                          numexpr =
         0;
23
24    VST_EXPRESSION_HANDLE exprh =
         (VST_EXPRESSION_HANDLE) NULL;
25    VST_CRITERIA_HANDLE criteriah =
         (VST_CRITERIA_HANDLE) NULL;
26
27    VST_COUNT field;
28    VST_MOUNT_CRITERIA_ORDER   sort;
29    VST_MEDIA_STAT_VALUE       value;
30    VST_MOUNT_CRITERIA_OPT     relopt;
31    VST_CONNECTIVE_OP          conop;
32
33    /* get the number of criteria within
         this group */
34    VS_CriteriaGroup_GetFields ( grouph,
35          VSID_NUMBER_ENTRIES, &numcrit,
36          VSID_ENDFIELD );
37
38    for ( i = 0 ; i < numcrit ; i++ )
39  {
40       /* get the criteria to print */
```

```
41       VS_CriteriaGroup_GetFields (
         grouph,
42          VSID_CRITERIA_HANDLE_ENTRY, i,
         &criteriah,
43          VSID_ENDFIELD );
44
45       /* get the criteria parameters */
46       VS_Criteria_GetFields (
         criteriah,
47          VSID_FIELD,
         &field,
48          VSID_MOUNT_CRITERIA_ORDER,
         &sort,
49          VSID_NUMBER_ENTRIES,
         &numexpr,
50          VSID_ENDFIELD );
51
52       printf ( "*** Criteria # %d
         ***\n", i );
53       printf ( " Field Index ==> %d\n",
         field );
54       printf ( " Mount Criteria Order
         ==> %d\n", sort );
55       printf ( "Number of Expressions
         ==> %d\n", numexpr );
56
57       for ( j = 0 ; j < numexpr ; j++ )
58       {
59          /* get the expression to print
         */
60          VS_Criteria_GetFields (
         criteriah,
61
         VSID_EXPRESSION_HANDLE_ENTRY, j,
         &exprh,
62               VSID_ENDFIELD );
63
64          /* get the expression's
         parameters */
65          VS_Expression_GetFields (
         exprh,
```

```
66              VSID_MOUNT_CRITERIA_OPT,
        &relopt,
67              VSID_CONNECTIVE_OP,
        &conop,
68              VSID_MEDIA_STAT_VALUE,
        value,
69              VSID_ENDFIELD );
70
71           printf ( "*** Expression # %d
        ***\n", j );
72           printf ( "Mount Criteria
        Option ==> %d\n", relopt);
73           printf ( " Media Stat Value ==>
        %s\n", value);
74           printf ( " Connective
        Operation ==> %d\n", conop);
75           }
76     }
77
78     return;
79 }
```


Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Criteria_Create(l),
- VS_Criteria_Destroy(l),
- VS_Criteria_GetFields(l),
- VS_Criteria_SetFields(l),
- VS_CriteriaGroup_Create(l),

- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_GetFields(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Expression_Create(l),
- VS_Expression_Destroy(l),
- VS_Expression_SetFields(l),
- VSCMD_Mount(l)

VS_ Expression_ SetFields

`VS_Expression_SetFields` sets the value of one or more fields in an expression handle. An expression handle is used to pass information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Expression_SetFields
(VST_EXPRESSION_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The expression handle where information is stored.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_MOUNT_CRITERIA_OP (VST_MOUNT_CRITERIA_OPT)	Relational operation for this expression:
VSID_CONNECTIVE_OP (VST_CONNECTIVE_OP)	Connective operation between two expressions. Valid `VSID_CONNECTIVE_OP` values are enumerated in the *`vs_types.h`* file. The last expression in a criteria must have the connect operator set to NONE.

Parameter Type	Description
VSID_MEDIA_STAT_VALUE (VST_MEDIA_STAT_VALUE)	Value for comparison. All values are compared as strings. If numbers are needed, they are left filled with zeros within the string.

Return Values

`VS_Expression_SetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - The appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not an expression handle.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_criteria
4  *
5  * PURPOSE:
6  * This function creates the mount
       criteria group
7  * handle andsets the values in it
       according to
```

```
8  * user input.
9  *
10 * PARAMETERS:
11 * none
12 *
13 ************************************
       *********/
14 #ifdef ANSI_C
15    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria(void)
16 #else
17    VST_CRITERIAGROUP_HANDLE
         vst_create_mount_criteria()
18 #endif
19 {
20    int                        i;
21    int                        j;
22    int                        numcrit;
23    int                        numexpr;
24    VST_BOOLEAN                rc =
         VSE_TRUE;
25    VST_EXPRESSION_HANDLE      exprh;
26    VST_CRITERIA_HANDLE
         criteriah;
27    VST_CRITERIAGROUP_HANDLE   grouph;
28    VST_COUNT                  field;
29    VST_MOUNT_CRITERIA_ORDER   sort;
30    VST_MEDIA_STAT_VALUE       value;
31    VST_MOUNT_CRITERIA_OPT     relopt;
32    VST_CONNECTIVE_OP          conop;
33
34    /* create the criteria group */
35    grouph = VS_CriteriaGroup_Create();
36
37    if ( grouph ==
         (VST_CRITERIAGROUP_HANDLE) NULL )
38    {
39       /* out of memory -- return */
40       return (
         (VST_CRITERIAGROUP_HANDLE) NULL
         );
41    }
```

```
42
43    /* populate the criteria group with
         criteria */
44    /* (upto 5) */
45    printf ( “Enter number of Criteria in
         group ==> “ );
46    numcrit = atoi(gets(input));
47
48    for ( i = 0 ; i < numcrit ; i++ )
49    {
50       /* create the criteria for a media
         stat field */
51       criteriah = VS_Criteria_Create();
52
53       if ( criteriah ==
         (VST_CRITERIA_HANDLE) NULL )
54       {
55          /* could not allocate handle */
56          rc = VSE_FALSE;
57          break;
58       }
59
60       printf ( “Enter the media’s field
         number ==> “ );
61       field = atoi(gets(input));
62
63       printf ( “Enter the sort order
         (Ascending - 1, Descending - 2)
         ==> “ );
64       sort = atoi(gets(input));
65
66       /* set the criteria parameters */
67       /VS_Criteria_SetFields (
         criteriah,
68                   VSID_FIELD,
         field,
69
         VSID_MOUNT_CRITERIA_ORDER, sort,
70                   VSID_ENDFIELD );
71
72       /* populate the critera with
         expressions */
```

```
73       /* (upto 4) */
74       printf ( “Enter the number of
         criteria expressions ==> “ );
75       numexpr = atoi(gets(input));
76
77       for ( j = 0 ; j < numexpr ; j++ )
78       {
79          /* create an expression for
         this */
80          /* criteria */
81          exprh =
         VS_Expression_Create();
82
83          if ( exprh ==
         (VST_EXPRESSION_HANDLE) NULL )
84          {
85             /* could not allocate memory
         for this */
86             /* handle */
87             rc = VSE_FALSE;
88             break;
89          }
90
91          printf ( “Enter relational
         option (eq 1, gt 2, ge 3, lt 4, le
         5, ne 6) ==> “ );
92          relopt = atoi(gets(input));
93
94          printf ( “Enter the media field
         value ==> “ );
95          gets( value);
96
97          printf ( “Enter connective
         operation (none 0, and 1, or 2)
         ==> “ );
98          conop = atoi(gets(input));
99
100         /* set the expression’s
         parameters */
101         VS_Expression_SetFields (
         exprh,
```

```
102           VSID_MOUNT_CRITERIA_OPT,
        relopt,
103           VSID_CONNECTIVE_OP,
        conop,
104           VSID_MEDIA_STAT_VALUE,
        value,
105           VSID_ENDFIELD );
106
107        /* add the expression to the
        criteria */
108        VS_Criteria_SetFields (
        criteriah,
109
        VSID_EXPRESSION_HANDLE_ENTRY, j,
        exprh,
110           VSID_ENDFIELD );
111
112     }
113     /* add the criteria to the
        criteria group */
114     VS_CriteriaGroup_SetFields (
        grouph,
115        VSID_CRITERIA_HANDLE_ENTRY, i,
        criteriah,
116        VSID_ENDFIELD );
117  }
118
119  /* if it failed, destroy the criteria
        group handle */
120  if ( rc == VSE_FALSE )
121  {
122     /* criteria group will destroy any
        */
123     /* criteria and their expressions
        */
124     /* for us */
125     VS_CriteriaGroup_Destroy ( grouph
        );
126
127     grouph =
        (VST_CRITERIAGROUP_HANDLE) NULL;
128  }
```

```
129
130   return ( grouph );
131}
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l), VS_Criteria_Create(l),
- VS_Criteria_Destroy(l),
- VS_Criteria_GetFields(l),
- VS_Criteria_SetFields(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_GetFields(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Expression__Create(l),
- VS_Expression_Destroy(l),
- VS_Expression_GetFields(l),
- VSCMD_Mount(l).

VS_Global_GetFields

`VS_Global_GetFields` retrieves global default parameters for all commands. A global handle is used to maintain global default parameter values for VolServ commands.

Tip

Global defaults can be overridden by command-level specific defaults. Command-specific defaults, in turn, can be overridden by parameter values specified in an actual command's parameter list.

Synopsis

```
VST_BOOLEAN VS_Global_GetFields
(
"…",
VSID_ENDFIELD )
```

Arguments

- `"…"` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ASYNC_PROGRAM_NUMBER (VST_PROGRAM_NUMBER *)	Pointer to the RPC program number to use for asynchronous processing.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH *)	Pointer to the dispatch function for all commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID *)	Pointer to the identifier of the enterprise, if any, to receive intermediate and final status on every command request.
VSID_INIT (int *)	Pointer to a flag that indicates whether the API is initialized. The value is VSE_TRUE if the API is initialized and VSE_FALSE otherwise.
VSID_NOTIFY_DISPATCH (VST_NOTIFY_DISPATCH *)	Pointer to the dispatch function used for notification (MediaClass callback) processing.
VSID_PRIORITY (VST_PRIORITY *)	Pointer to the default execution priority to be assigned to every command request. Default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT *)	Pointer to the default number of times the API software retries for command status from VolServ before returning a time-out to the client software for every command request. Total length of time the API software waits for a command status from VolServ is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. Default value is 3.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG*)	Pointer to a flag that indicates whether the API software is to wait for final status from VolServ (or to time-out) for a request. Valid options are VSE_TRUE (API is to wait for final status) and VSE_FALSE (API is not to wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). Default value is VSE_TRUE.
VSID_SYNC_PROGRAM_NUMBER (VST_PROGRAM_NUMBER *)	Pointer to the RPC program number to use for synchronous processing.
VSID_TIMEOUT_VALUE (VST_TIME_OUT *)	Pointer to the amount of time (in seconds) the API software is to wait for status from VolServ before returning a time-out to the client software. Default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Pointer to the default value to be placed in the VSID_USER_FIELD for every command request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for each command. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VS_Global_GetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - The appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_show_globals
4  *
5  * PURPOSE:
6  * This function allows the user to
       display the
7  * VolServ API's global parameters.
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_show_globals( void )
14 #else
15    VST_BOOLEAN vst_show_globals()
16 #endif
17 {
18    VST_PRIORITY            priority;
19    VST_USER_FIELD          user_field;
20    VST_TIME_OUT            timeout;
21    VST_RETRY_LIMIT         retries;
22    VST_STATUS_WAIT_FLAG    wait;
23    VST_ENTERPRISE_ID       enterprise;
24
25    VS_Global_GetFields(VSID_PRIORITY,
         &priority,
26             VSID_USER_FIELD,
         user_field,
27             VSID_TIMEOUT_VALUE,
         &timeout,
28             VSID_RETRY_LIMIT,
         &retries,
29             VSID_STATUS_WAIT_FLAG,
         &wait,
30             VSID_ENTERPRISE_ID,
         &enterprise,
31             VSID_ENDFIELD);
32    printf("*** Global Parameters
         ***\n");
```

```
33    printf("Priority = %d User field =
         [%s]\n", priority, user_field);
34    printf("Time out value = %d Retry
         count = %d\n", timeout, retries);
35    printf("Status wait = %d Enterprise =
         %lu\n", wait, enterprise);
36 }
```

Notes

If a string that is passed to hold the user field does not have enough space allocated, unpredictable results may occur.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on command requests submitted through the API interface to the VolServ system.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VS_Select(l)

VS_Global_SetFields

`VS_Global_SetFields` sets the value of one or more fields in a global handle. A global handle is used to maintain global default parameter values for VolServ commands.

Tip

Global defaults can be overridden by command-level specific defaults. Command-specific defaults, in turn, can be overridden by parameter values specified in an actual command's parameter list.

Synopsis

```
VST_BOOLEAN VS_Global_SetFields
(
"…",
VSID_ENDFIELD )
```

Arguments

- `"…"` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ASYNC_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	RPC program number to use for asynchronous processing. If not specified or 0, the API uses a transient number. Default value is 0.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Dispatch function for all commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on every command request.
VSID_NOTIFY_DISPATCH (VST_NOTIFY_DISPATCH)	Pointer to the dispatch function used for notification (MediaClass callback) processing.
VSID_PRIORITY (VST_PRIORITY)	Default execution priority to be assigned to every command. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. Default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Default number of times the API software is to retry for command status from VolServ before returning a time-out to the client software for every command request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode. Default value is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software is to wait for final status from VolServ (or to time-out) for a command. Valid options are `VSE_TRUE` (API is to wait for final status) and `VSE_FALSE` (API is not to wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). Default value is `VSE_TRUE`.

Parameter Type	Description
VSID_SYNC_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	RPC program number to use for synchronous processing. If not specified or 0, the API uses a transient number. Default value is 0.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. Default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Pointer to the default value to be put in the VSID_USER_FIELD for every command. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for each command. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VS_Global_SetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - The appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not an expression handle.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_set_globals
4  *
5  * PURPOSE:
6  * This function allows the user to set
       VolServ API's
7  * global parameters.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN vst_set_globals( void )
15 #else
16    VST_BOOLEAN vst_set_globals()
17 #endif
18 {
19    VST_PRIORITY priority;
20    VST_USER_FIELD user_field;
21    VST_TIME_OUT timeout;
22    VST_RETRY_LIMIT retries;
23    VST_STATUS_WAIT_FLAG wait_flag;
24    VST_ENTERPRISE_ID enterprise_id;
25    VST_BOOLEAN rc;
26
27    printf("*** Set Global Parameters
         ***\n\n");
28    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
29    rc =
         VS_Global_SetFields(VSID_PRIORITY
         , priority,
```

```
30              VSID_USER_FIELD,
          user_field,
31              VSID_TIMEOUT_VALUE,
          timeout,
32              VSID_RETRY_LIMIT,
          retries,
33              VSID_STATUS_WAIT_FLAG,
          wait_flag,
34              VSID_ENTERPRISE_ID,
          enterprise_id,
35              VSID_ENDFIELD);
36     return (rc);
37 }
```

Notes

Total length of time the API software waits for a command status from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

This function can be called before `VS_Initialize`. To set the SYNC/ASYNC program numbers, this function must be called before `VS_Initialize`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on command requests submitted through the API interface to the VolServ system.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_GetFields(l),
- VS_Select(l),

- VS_Initialize(l)

VS_Initialize

`VS_Initialize` readies the VolServ API library for use by the client software.

`VS_Initialize` creates and registers RPC transports for receiving VolServ status messages and also performs initialization processing required by other API functions.

`VS_Initialize` must be called before any API command functions are called.

Synopsis

```
VST_BOOLEAN VS_Initialize
( VST_HOSTNAME hostname,
VST_PROGRAM_NUMBER prognum,
int time-out )
```

Arguments

- `hostname` = Name of the computer running VolServ.
  - If NULL is specified, hostname defaults to the current host machine.
- `prognum` = Program number on which VolServ is registered.
  - If 0 is specified, prognum defaults to 300016 - the VolServ default.
- `timeout` = Amount of time, in seconds, to wait for an initial status from VolServ before timing out.

Return Values

`VS_Initialize` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - The appropriate error code is set in `VSG_Error.`
- `VSE_ERR_INITIALIZED` - VolServ API is already initialized.
- `VSE_ERR_OUTOFMEM` - Memory allocation call failed.
- `VSE_ERR_PMAPFAILED` - RPC registration for return status failed. RPC address specified could not be registered with the local machine's port mapper.
- `VSE_ERR_SYSTEMCALL` - A system call failed (usually from RPC). This generic error code covers an error that stems from a system call. The API sets this error code when encountering a failure during RPC setup.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_initialize
4  *
5  * PURPOSE:
6  * This function initializes the VolServ
       API.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_initialize(void)
14 #else
15    VST_BOOLEAN vst_initialize()
16 #endif
17 {
```

```
18    int                  timeout;
19    VST_HOSTNAME         host;
20    VST_PROGRAM_NUMBER   prognum;
21    VST_PROGRAM_NUMBER   syncnum;
22    VST_PROGRAM_NUMBER   asyncnum;
23    VST_BOOLEAN          rc = VSE_FALSE;
24
25    /* get parameters from user */
26    printf("*** Initialize parameters
         ***\n" );
27    printf("Enter VolServ Host ==> " );
28    gets( host );
29    printf("Enter VolServ Program Number
         ==> " );
30    prognum = atol(gets(input));
31    printf("Enter Timeout Value ==> " );
32    timeout = atoi(gets(input));
33
34    /* change the sync/async program
         numbers */
35    printf("*** RPC Program numbers
         ***\n" );
36    printf("Enter Sync Program Number (0
         for default) ==> " );
37    syncnum = atol(gets(input));
38    printf("Enter Async Program Number (0
         for default) ==> " );
39    asyncnum = atol(gets(input));
40
41    if ( syncnum != 0 )
42    {
43       VS_Global_SetFields (
44
         VSID_SYNC_PROGRAM_NUMBER,
         syncnum,
45                VSID_ENDFIELD );
46    }
47
48    if ( asyncnum != 0 )
49    {
50       VS_Global_SetFields (
```

```
51
        VSID_ASYNC_PROGRAM_NUMBER,
        asyncnum,
52              VSID_ENDFIELD );
53    }
54    rc = VS_Initialize(host, prognum,
        timeout);
55    return(rc);
56 }
```

Notes

`VS_Global_SetDefaults` must be called before `VS_Initialize` is called to set the SYNC/ASYNC program numbers.

All command functions fail if `VS_Initialize` is not invoked.

`VS_Initialize` does not verify that VolServ is currently running on the given host. Use `VS_Ping` to check VolServ's status.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Terminate(1),
- VS_Global_SetFields(l)

VS_Media_Create

`VS_Media_Create` allocates a VolServ API media handle. A media handle is used to pass media information to and from VolServ.

Synopsis

VST_MEDIA_HANDLE VS_Media_Create (void)

Arguments

None

Return Values

`VS_Media_Create` returns:

- A media handle, if one could be allocated.
- NULL, if a media handle could not be allocated. An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_media_handle
4  *
5  * PURPOSE:
6  * This function tests a media handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_media_handle(void)
14 #else
15    VST_BOOLEAN vst_media_handle()
16 #endif
17 {
```

```
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIA_HANDLE        h;
20    VST_MEDIA_ID            MediaID;
21    VST_MEDIA_TYPE_NAME
         MediaTypeName;
22    VST_BATCH_NAME          BatchName;
23    VST_MANUFACTURER_NAME
         Manufacturer;
24    VST_MEDIA_CLASS_NAME
         MediaClassName;
25    VST_MEDIA_LOC_STATE
         LocationState;
26    VST_ACTION_STATE        ActionState;
27    VST_ARCHIVE_NAME
         CurrentArchive;
28    VST_ARCHIVE_NAME
         PendingArchive;
29    VST_TIME                ImportDate;
30    VST_TIME
         LastDismount;
31    VST_ASSIGNMENT          Assignment;
32    int                     MountCount;
33    int                     MoveCount;
34
35    /* create the handle */
36    h = VS_Media_Create();
37    if (h != (VST_MEDIA_HANDLE) NULL)
38    {
39       /* get values from user */
40       printf("*** Media Handle ***\n");
41       printf("Enter media id ==> ");
42       gets(MediaID);
43       printf("Enter media type ==> ");
44       gets(MediaTypeName);
45       printf("Enter batch name ==> ");
46       gets(BatchName);
47       printf("Enter Manufacturer ==>
         ");
48       gets(Manufacturer);
49       printf("Enter Media Class Name ==>
         ");
```

```
50       gets(MediaClassName);
51       printf("Enter media location
         state ==> ");
52       LocationState =
         atoi(gets(input));
53       printf("Enter action state ==> ");
54       ActionState = atoi(gets(input));
55       printf("Enter current archive ==>
         ");
56       gets(CurrentArchive);
57       printf("Enter pending archive ==>
         ");
58       gets(PendingArchive);
59       printf("enter assignment value
         ==> ");
60       Assignment = atoi(gets(input));
61       printf("enter mount count ==> ");
62       MountCount = atoi(gets(input));
63       printf("enter move count ==> ");
64       MoveCount = atoi(gets(input));
65       /* set the fields in the handle */
66       rc = VS_Media_SetFields(h,
67          VSID_MEDIA_ID,
         MediaID,
68          VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
69          VSID_BATCH_NAME,
         BatchName,
70          VSID_MANUFACTURER,
         Manufacturer,
71          VSID_MEDIA_CLASS_NAME,
         MediaClassName,
72          VSID_MEDIA_LOC_STATE,
         LocationState,
73          VSID_ACTION_STATE,
         ActionState,
74          VSID_CURRENT_ARCHIVE_NAME,
         CurrentArchive,
75          VSID_PENDING_ARCHIVE_NAME,
         PendingArchive,
76          VSID_ASSIGNMENT,
         Assignment,
```

```
77          VSID_MOUNT_COUNT,
          MountCount,
78          VSID_MOVE_COUNT,
          MoveCount,
79          VSID_ENDFIELD);
80        if (rc)
81        {
82           vst_print_media(h);
83        }
84        VS_Media_Destroy(h);
85     }
86     return(rc);
87  }
```

Notes

None

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Media_Destroy(l),
- VS_Media_GetFields(l),
- VS_Media_SetFields(l)

VS_Media_Destroy

`VS_Media_Destroy` deallocates a VolServ API media handle that was allocated by `VS_Media_Create`. A media destroy handle is used to pass media information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Media_Destroy
(VST_MEDIA_HANDLE handle)

Arguments

- `handle` = The media handle to be destroyed.

Return Values

`VS_Media_Destroy` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADHANDLE` - Specified handle was not a media handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_media_handle
4  *
5  * PURPOSE:
6  * This function tests a media handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
```

```
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_media_handle(void)
14 #else
15    VST_BOOLEAN vst_media_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIA_HANDLE        h;
20    VST_MEDIA_ID            MediaID;
21    VST_MEDIA_TYPE_NAME
         MediaTypeName;
22    VST_BATCH_NAME          BatchName;
23    VST_MANUFACTURER_NAME
         Manufacturer;
24    VST_MEDIA_CLASS_NAME
         MediaClassName;
25    VST_MEDIA_LOC_STATE
         LocationState;
26    VST_ACTION_STATE        ActionState;
27    VST_ARCHIVE_NAME
         CurrentArchive;
28    VST_ARCHIVE_NAME
         PendingArchive;
29    VST_TIME                ImportDate;
30    VST_TIME
         LastDismount;
31    VST_ASSIGNMENT          Assignment;
32    int                     MountCount;
33    int                     MoveCount;
34
35    /* create the handle */
36    h = VS_Media_Create();
37    if (h != (VST_MEDIA_HANDLE) NULL)
38    {
39       /* get values from user */
40       printf("*** Media Handle ***\n");
41       printf("Enter media id ==> ");
42       gets(MediaID);
43       printf("Enter media type ==> ");
44       gets(MediaTypeName);
45       printf("Enter batch name ==> ");
```

```
46       gets(BatchName);
47       printf("Enter Manufacturer ==>
         ");
48       gets(Manufacturer);
49       printf("Enter Media Class Name ==>
         ");
50       gets(MediaClassName);
51       printf("Enter media location
         state ==> ");
52       LocationState =
         atoi(gets(input));
53       printf("Enter action state ==> ");
54       ActionState = atoi(gets(input));
55       printf("Enter current archive ==>
         ");
56       gets(CurrentArchive);
57       printf("Enter pending archive ==>
         ");
58       gets(PendingArchive);
59       printf("enter assignment value
         ==> ");
60       Assignment = atoi(gets(input));
61       printf("enter mount count ==> ");
62       MountCount = atoi(gets(input));
63       printf("enter move count ==> ");
64       MoveCount = atoi(gets(input));
65       /* set the fields in the handle */
66       rc = VS_Media_SetFields(h,
67          VSID_MEDIA_ID,
         MediaID,
68          VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
69          VSID_BATCH_NAME,
         BatchName,
70          VSID_MANUFACTURER,
         Manufacturer,
71          VSID_MEDIA_CLASS_NAME,
         MediaClassName,
72          VSID_MEDIA_LOC_STATE,
         LocationState,
73          VSID_ACTION_STATE,
         ActionState,
```

```
74          VSID_CURRENT_ARCHIVE_NAME,
          CurrentArchive,
75          VSID_PENDING_ARCHIVE_NAME,
          PendingArchive,
76          VSID_ASSIGNMENT,
          Assignment,
77          VSID_MOUNT_COUNT,
          MountCount,
78          VSID_MOVE_COUNT,
          MoveCount,
79          VSID_ENDFIELD);
80        if (rc)
81        {
82          vst_print_media(h);
83        }
84        VS_Media_Destroy(h);
85     }
86     return(rc);
87 }
```

Notes

After `VS_Media_Destroy` has been called for a media handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Media_Create(l),
- VS_Media_GetFields(l),
- VS_Media_SetFields(l)

VS_Media_GetFields

`VS_Media_GetFields` retrieves information associated with a media handle. A media handle is used to pass media information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Media_GetFields
(VST_MEDIA_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The media handle where information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ACTION_STATE (VST_ACTION_STATE *)	Action state associated with this medium. Valid `VSID_ACTION_STATE` values are enumerated in the *vs_types.h* file.
VSID_ASSIGNMENT (VST_ASSIGNMENT *)	Current assignment of this medium. Valid `VSID_ASSIGNMENT` values are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_BATCH_NAME (VST_BATCH_NAME)	Pointer to the name of the batch associated with this medium. Valid batch names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CURRENT_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Pointer to the name of the archive where the medium is currently stored. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_IMPORT_DATE (VST_TIME *)	Pointer to the date this medium was imported into the VolServ system.
VSID_LAST_DISMOUNT (VST_TIME *)	Pointer to the time this medium was last dismounted.
VSID_MANUFACTURER (VST_MANUFACTURER_NAME)	Pointer to the manufacturer associated with this medium. Valid manufacturer names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Pointer to the MediaClass group with which this medium is associated.
VSID_MEDIA_ID (VST_MEDIA_ID)	Pointer to the identifier of this medium.
VSID_MEDIA_LOC_STATE (VST_MEDIA_LOC_STATE *)	Pointer to the location state of this medium. Valid `VSID_MEDIA_LOC_STATE` values are enumerated in the *vs_types.h* file
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	Pointer to the media type associated with this medium. Valid media type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MOUNT_COUNT (VST_COUNT *)	Pointer to the number of times this medium has been mounted.

Parameter Type	Description
VSID_MOVE_COUNT (VST_COUNT *)	Pointer to the number of times this medium has been moved from one archive to another.
VSID_PENDING_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Pointer to the name of the destination archive for this medium. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Return Values

`VS_Media_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a media handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_media
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored
7  * in a media handle.
8  * PARAMETERS:
9  * h : the media handle to print
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13 void vst_print_media(VST_MEDIA_HANDLE
       h)
14 #else
15 void vst_print_media(h)
16 VST_MEDIA_HANDLE h;
17 #endif
18 {
19    VST_MEDIA_ID            MediaID;
20    VST_MEDIA_TYPE_NAME
         MediaTypeName;
21    VST_BATCH_NAME          BatchName;
22    VST_MANUFACTURER_NAME
         Manufacturer;
23    VST_MEDIA_CLASS_NAME
         MediaClassName;
24    VST_MEDIA_LOC_STATE
         LocationState;
25    VST_ACTION_STATE        ActionState;
26    VST_ARCHIVE_NAME
         CurrentArchive;
27    VST_ARCHIVE_NAME
         PendingArchive;
28    VST_TIME                ImportDate;
29    VST_TIME
         LastDismount;
30    VST_ASSIGNMENT          Assignment;
31    int                     MountCount;
32    int                     MoveCount;
33
34          VS_Media_GetFields(h,
35          VSID_MEDIA_ID,
         MediaID,
36          VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
37          VSID_BATCH_NAME,
         BatchName,
38          VSID_MANUFACTURER,
         Manufacturer,
```

```
39          VSID_MEDIA_CLASS_NAME,
          MediaClassName,
40          VSID_MEDIA_LOC_STATE,
          &LocationState,
41          VSID_ACTION_STATE,
          &ActionState,
42          VSID_CURRENT_ARCHIVE_NAME,
          CurrentArchive,
43          VSID_PENDING_ARCHIVE_NAME,
          PendingArchive,
44          VSID_IMPORT_DATE,
          &ImportDate,
45          VSID_LAST_DISMOUNT,
          &LastDismount,
46          VSID_ASSIGNMENT,
          &Assignment,
47          VSID_MOUNT_COUNT,
          &MountCount,
48          VSID_MOVE_COUNT,
          &MoveCount,
49          VSID_ENDFIELD);
50    printf("******Media
          Handle******\n");
51    printf("Media ID = %s\n",MediaID);
52    printf("Media Type Name =
          %s\n",MediaTypeName);
53    printf("Batch Name =
          %s\n",BatchName);
54    printf("Manufacturer =
          %s\n",Manufacturer);
55    printf("Media Class Name =
          %s\n",MediaClassName);
56    printf("Media Loc State =
          %d\n",LocationState);
57    printf("Action State =
          %d\n",ActionState);
58    printf("Current Archive =
          %s\n",CurrentArchive);
59    printf("Pending Archive =
          %s\n",PendingArchive);
60    printf("Import Date =
          %d\n",ImportDate);
```

```
61    printf("Last Dismount =
         %d\n",LastDismount);
62    printf("Assignment =
         %d\n",Assignment);
63    printf("Move Count =
         %d\n",MoveCount);
64    printf("Mount Count =
         %d\n",MountCount);
65 }
```

Notes

The `VSID_IMPORT_DATE` and last `VSID_LAST_DISMOUNT` are kept as long integers; use the **ctime** function to convert either of these values to a string.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Media_Create(l),
- VS_Media_Destroy(l),
- VS_Media_SetFields(l),
- VSCMS_MediaQuery(l),
- VSCMD_MediaClassQuery(l)

VS_Media_SetFields

`VS_Media_SetFields` sets the value of one or more fields associated with a media handle. A media handle is used to pass media information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Media_SetFields
(VST_MEDIA_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The media handle where information is stored or updated.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ACTION_STATE (VST_ACTION_STATE)	Action state associated with this medium. Valid `VSID_ACTION_STATE` values are enumerated in the *vs_types.h* file.
VSID_ASSIGNMENT (VST_ASSIGNMENT)	Current assignment of this medium. Valid `VSID_ASSIGNMENT` values are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_BATCH_NAME (VST_BATCH_NAME)	Name of the batch associated with this medium. Valid batch names may contain 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CURRENT_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive where the medium is currently stored. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_IMPORT_DATE (VST_TIME)	Date this medium was imported into the VolServ system.
VSID_LAST_DISMOUNT (VST_TIME)	Time this medium was last dismounted.
VSID_MANUFACTURER (VST_MANUFACTURER_NAME)	Manufacturer associated with this medium. Valid manufacturer names may contain 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	MediaClass group with which this medium is associated.
VSID_MEDIA_ID (VST_MEDIA_ID)	Identifier of this medium.
VSID_MEDIA_LOC_STATE (VST_MEDIA_LOC_STATE)	Location state of this medium. Valid VSID_MEDIA_LOC_STATE values are enumerated in the *vs_types.h* file
VSID_MEDIA_TYPE_NAM (VST_MEDIA_TYPE_NAME)	Media type associated with this medium. Valid media types may contain 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MOUNT_COUNT (VST_COUNT)	Number of times this medium has been mounted.
VSID_MOVE_COUNT (VST_COUNT)	Number of times this medium has been moved from one archive to another.

Parameter Type	Description
VSID_PENDING_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the destination archive for this medium. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Return Values

`VS_Media_SetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a media handle.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_media_handle
4  *
5  * PURPOSE:
6  * This function tests a media handle.
7  *
8  * PARAMETERS:
9  * none
```

```
10 *
11 ************************************
        *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_media_handle(void)
14 #else
15    VST_BOOLEAN vst_media_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIA_HANDLE        h;
20    VST_MEDIA_ID            MediaID;
21    VST_MEDIA_TYPE_NAME
         MediaTypeName;
22    VST_BATCH_NAME          BatchName;
23    VST_MANUFACTURER_NAME
         Manufacturer;
24    VST_MEDIA_CLASS_NAME
         MediaClassName;
25    VST_MEDIA_LOC_STATE
         LocationState;
26    VST_ACTION_STATE        ActionState;
27    VST_ARCHIVE_NAME
         CurrentArchive;
28    VST_ARCHIVE_NAME
         PendingArchive;
29    VST_TIME                ImportDate;
30    VST_TIME
         LastDismount;
31    VST_ASSIGNMENT          Assignment;
32    int                     MountCount;
33    int                     MoveCount;
34
35    /* create the handle */
36    h = VS_Media_Create();
37    if (h != (VST_MEDIA_HANDLE) NULL)
38    {
39       /* get values from user */
40       printf("*** Media Handle ***\n");
41       printf("Enter media id ==> ");
42       gets(MediaID);
```

```
43        printf("Enter media type ==> ");
44        gets(MediaTypeName);
45        printf("Enter batch name ==> ");
46        gets(BatchName);
47        printf("Enter Manufacturer ==>
          ");
48        gets(Manufacturer);
49        printf("Enter Media Class Name ==>
          ");
50        gets(MediaClassName);
51        printf("Enter media location
          state ==> ");
52        LocationState =
          atoi(gets(input));
53        printf("Enter action state ==> ");
54        ActionState = atoi(gets(input));
55        printf("Enter current archive ==>
          ");
56        gets(CurrentArchive);
57        printf("Enter pending archive ==>
          ");
58        gets(PendingArchive);
59        printf("enter assignment value
          ==> ");
60        Assignment = atoi(gets(input));
61        printf("enter mount count ==> ");
62        MountCount = atoi(gets(input));
63        printf("enter move count ==> ");
64        MoveCount = atoi(gets(input));
65        /* set the fields in the handle */
66        rc = VS_Media_SetFields(h,
67           VSID_MEDIA_ID,
          MediaID,
68           VSID_MEDIA_TYPE_NAME,
          MediaTypeName,
69           VSID_BATCH_NAME,
          BatchName,
70           VSID_MANUFACTURER,
          Manufacturer,
71           VSID_MEDIA_CLASS_NAME,
          MediaClassName,
```

```
72          VSID_MEDIA_LOC_STATE,
          LocationState,
73          VSID_ACTION_STATE,
          ActionState,
74          VSID_CURRENT_ARCHIVE_NAME,
          CurrentArchive,
75          VSID_PENDING_ARCHIVE_NAME,
          PendingArchive,
76          VSID_ASSIGNMENT,
          Assignment,
77          VSID_MOUNT_COUNT,
          MountCount,
78          VSID_MOVE_COUNT,
          MoveCount,
79          VSID_ENDFIELD);
80        if (rc)
81        {
82          vst_print_media(h);
83        }
84        VS_Media_Destroy(h);
85     }
86     return(rc);
87 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Media_Create(l),
- VS_Media_Destroy(l),
- VS_Media_SetFields(l)

VS_ MediaClass_ Create

`VS_MediaClass_Create` allocates a VolServ API MediaClass handle. A MediaClass handle is used to pass MediaClass information to and from VolServ.

Synopsis

VST_MEDIACLASS_HANDLE VS_MediaClass_Create (void)

Arguments

None

Return Values

`VS_MediaClass_Create` returns:

- A MediaClass handle, if one can be allocated.
- NULL, if a MediaClass handle could not be allocated. An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediaclass_handle
4  *
5  * PURPOSE:
6  * This function tests a mediaclass
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_mediaclass_handle(void)
```

```
14 #else
15    VST_BOOLEAN vst_mediaclass_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIACLASS_HANDLE   h;
20    VST_MEDIA_CLASS_NAME    MediaClass;
21    VST_MEDIA_TYPE_NAME
         MediaTypeName;
22    VST_PRIORITY
         ReleasePriority;
23    VST_CAPACITY            Capacity;
24    VST_FILL_LEVEL          FillLevel;
25    VST_CLASS_MOUNT_STATE   MountState;
26    VST_HIGH_MARK           HighMark;
27    VST_CLASS_RPC_OPTION    RPC_Option;
28    VST_HOSTNAME
         RPC_HostName;
29    VST_PROGRAM_NUMBER      RPC_ProgNum;
30    VST_VERSION_NUMBER      RPC_VersNum;
31    VST_PROCEDURE_NUMBER    RPC_ProcNum;
32    VST_PROTOCOL
         RPC_Protocol;
33    VST_ENTERPRISE_ID
         EnterpriseID;
34    VST_NOTIFY_COMMENT
         NotifyComment;
35
36    /* create the handle */
37    h = VS_MediaClass_Create();
38    if (h != (VST_MEDIACLASS_HANDLE)
         NULL)
39    {
40       /* get values from user */
41       printf("*** Media Class Handle
         ***\n");
42       printf("Enter mediaclass name ==>
         ");
43       gets(MediaClass);
44       printf("Enter media type name ==>
         ");
```

```
45       gets(MediaTypeName);
46       printf("Enter release priority
         ==> ");
47       ReleasePriority =
         atoi(gets(input));
48       printf("Enter capacity ==> ");
49       Capacity = atoi(gets(input));
50       printf("Enter fill level ==> ");
51       FillLevel = atoi(gets(input));
52       printf("Enter mount state ==> ");
53       MountState = atoi(gets(input));
54       printf("Enter high mark ==> ");
55       HighMark = atoi(gets(input));
56       printf("Enter class RPC option ==>
         ");
57       RPC_Option = atoi(gets(input));
58       printf("Enter RPC host name ==>
         ");
59       gets(RPC_HostName);
60       printf("Enter RPC program number
         ==> ");
61       RPC_ProgNum = atol(gets(input));
62       printf("Enter RPC version number
         ==> ");
63       RPC_VersNum = atol(gets(input));
64       printf("Enter RPC procedure
         number ==> ");
65       RPC_ProcNum = atol(gets(input));
66       printf("Enter RPC protocol ==> ");
67       RPC_Protocol = atoi(gets(input));
68       printf("Enter enterprise id ==>
         ");
69       EnterpriseID = atol(gets(input));
70       printf("Enter notify comment ==>
         ");
71       gets(NotifyComment);
72       /* set the fields */
73       rc = VS_MediaClass_SetFields(h,
74          VSID_MEDIA_CLASS_NAME,
         MediaClass,
75          VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
```

```
76          VSID_RELEASE_PRIORITY,
          ReleasePriority,
77          VSID_CAPACITY,
          Capacity,
78          VSID_FILL_LEVEL,
          FillLevel,
79          VSID_CLASS_MOUNT_STATE,
          MountState,
80          VSID_HIGH_MARK,
          HighMark,
81          VSID_CLASS_RPC_OPTION,
          RPC_Option,
82          VSID_HOST_NAME,
          RPC_HostName,
83          VSID_PROGRAM_NUMBER,
          RPC_ProgNum,
84          VSID_VERSION_NUMBER,
          RPC_VersNum,
85          VSID_PROCEDURE_NUMBER,
          RPC_ProcNum,
86          VSID_PROTOCOL,
          RPC_Protocol,
87          VSID_ENTERPRISE_ID,
          EnterpriseID,
88          VSID_NOTIFY_COMMENT,
          NotifyComment,
89          VSID_ENDFIELD);
90       if (rc)
91       {
92          vst_print_mediaclass(h);
93       }
94       VS_MediaClass_Destroy(h);
95    }
96    return(rc);
97 }
```

Notes None

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_MediaClass_Destroy(l),
- VS_MediaClass_GetFields(l),
- VS_MediaClass_SetFields(l)

VS_ MediaClass_ Destroy

VS_MediaClass_Destroy deallocates a VolServ API MediaClass handle that was allocated with VS_MediaClass_Create. A MediaClass handle is used to pass MediaClass information to and from VolServ.

Synopsis

VST_BOOLEAN VS_MediaClass_Destroy
(VST_MEDIACLASS_HANDLE handle)

Arguments

- handle = The MediaClass handle to be destroyed.

Return Values

VS_MediaClass_Destroy returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a MediaClass handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediaclass_handle
4  *
5  * PURPOSE:
6  * This function tests a mediaclass
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_mediaclass_handle(void)
14 #else
15    VST_BOOLEAN vst_mediaclass_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIACLASS_HANDLE   h;
20    VST_MEDIA_CLASS_NAME    MediaClass;
21    VST_MEDIA_TYPE_NAME
         MediaTypeName;
22    VST_PRIORITY
         ReleasePriority;
23    VST_CAPACITY            Capacity;
24    VST_FILL_LEVEL          FillLevel;
25    VST_CLASS_MOUNT_STATE   MountState;
26    VST_HIGH_MARK           HighMark;
27    VST_CLASS_RPC_OPTION    RPC_Option;
28    VST_HOSTNAME
         RPC_HostName;
29    VST_PROGRAM_NUMBER      RPC_ProgNum;
30    VST_VERSION_NUMBER      RPC_VersNum;
31    VST_PROCEDURE_NUMBER    RPC_ProcNum;
32    VST_PROTOCOL
         RPC_Protocol;
33    VST_ENTERPRISE_ID
         EnterpriseID;
34    VST_NOTIFY_COMMENT
         NotifyComment;
35
36    /* create the handle */
37    h = VS_MediaClass_Create();
38    if (h != (VST_MEDIACLASS_HANDLE)
         NULL)
39    {
40       /* get values from user */
41       printf("*** Media Class Handle
         ***\n");
```

```
42      printf("Enter mediaclass name ==>
        ");
43      gets(MediaClass);
44      printf("Enter media type name ==>
        ");
45      gets(MediaTypeName);
46      printf("Enter release priority
        ==> ");
47      ReleasePriority =
        atoi(gets(input));
48      printf("Enter capacity ==> ");
49      Capacity = atoi(gets(input));
50      printf("Enter fill level ==> ");
51      FillLevel = atoi(gets(input));
52      printf("Enter mount state ==> ");
53      MountState = atoi(gets(input));
54      printf("Enter high mark ==> ");
55      HighMark = atoi(gets(input));
56      printf("Enter class RPC option ==>
        ");
57      RPC_Option = atoi(gets(input));
58      printf("Enter RPC host name ==>
        ");
59      gets(RPC_HostName);
60      printf("Enter RPC program number
        ==> ");
61      RPC_ProgNum = atol(gets(input));
62      printf("Enter RPC version number
        ==> ");
63      RPC_VersNum = atol(gets(input));
64      printf("Enter RPC procedure
        number ==> ");
65      RPC_ProcNum = atol(gets(input));
66      printf("Enter RPC protocol ==> ");
67      RPC_Protocol = atoi(gets(input));
68      printf("Enter enterprise id ==>
        ");
69      EnterpriseID = atol(gets(input));
70      printf("Enter notify comment ==>
        ");
71      gets(NotifyComment);
72      /* set the fields */
```

```
73       rc = VS_MediaClass_SetFields(h,
74          VSID_MEDIA_CLASS_NAME,
         MediaClass,
75          VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
76          VSID_RELEASE_PRIORITY,
         ReleasePriority,
77          VSID_CAPACITY,
         Capacity,
78          VSID_FILL_LEVEL,
         FillLevel,
79          VSID_CLASS_MOUNT_STATE,
         MountState,
80          VSID_HIGH_MARK,
         HighMark,
81          VSID_CLASS_RPC_OPTION,
         RPC_Option,
82          VSID_HOST_NAME,
         RPC_HostName,
83          VSID_PROGRAM_NUMBER,
         RPC_ProgNum,
84          VSID_VERSION_NUMBER,
         RPC_VersNum,
85          VSID_PROCEDURE_NUMBER,
         RPC_ProcNum,
86          VSID_PROTOCOL,
         RPC_Protocol,
87          VSID_ENTERPRISE_ID,
         EnterpriseID,
88          VSID_NOTIFY_COMMENT,
         NotifyComment,
89          VSID_ENDFIELD);
90       if (rc)
91       {
92          vst_print_mediaclass(h);
93       }
94       VS_MediaClass_Destroy(h);
95    }
96    return(rc);
97 }
```

Notes

After `VS_MediaClass_Destroy` has been called for a MediaClass handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_MediaClass_Create(l),
- VS_MediaClass_GetFields(l),
- VS_MediaClass_SetFields(l)

VS_ MediaClass_ GetFields

`VS_MediaClass_GetFields` retrieves information associated with a MediaClass handle. A MediaClass handle is used to pass MediaClass information to and from VolServ.

Synopsis

VST_BOOLEAN VS_MediaClass_GetFields
(VST_MEDIACLASS_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The MediaClass handle where information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CAPACITY (VST_CAPACITY *)	Pointer to the maximum number of media allowed in this MediaClass group.
VSID_CLASS_MOUNT_STATE (VST_CLASS_MOUNT_STATE *)	Pointer that indicates whether this MediaClass group supports the "mount by MediaClass" functionality. Valid `VSID_CLASS_MOUNT_STATE` values are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_CLASS_RPC_OPTION (VST_CLASS_RPC_OPTION *)	Pointer that indicates whether callbacks are to be activated for this MediaClass group and if they are, which callback scheme is to be used. Valid `VSID_CLASS_RPC_OPTION` values are enumerated in the *vs_types.h* file.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID *)	If `VSID_CLASS_RPC_OPTION` is set to `VSE_CLASS_RPC_ENTERPRISE`, a pointer to the identifier of the enterprise to receive unsolicited callbacks. Otherwise, `VSID_ENTERPRISE_ID` is not applicable.
VSID_FILL_LEVEL (VST_FILL_LEVEL *)	Pointer to the current number of media in this MediaClass group.
VSID_HIGH_MARK (VST_HIGH_MARK *)	Pointer to the percentage of the MediaClass capacity above which notification or automatic media migration is initiated.
VSID_HOST_NAME (VST_HOSTNAME)	If `VSID_CLASS_RPC_OPTION` is set to `VSE_CLASS_RPC_STANDARD`, a pointer to the network-assigned name of the computer where the task that “listens” for unsolicited callbacks executes. Otherwise, `VSID_HOST_NAME` is not applicable.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Pointer to the name of this MediaClass group.
VSID_MEDIA_HANDLE_TABLE (VST_TABLE_HANDLE *)	Media (in table format) that belong to this MediaClass group.
VSID_MEDIA_ID (VST_MEDIA_ID)	Pointer to the first media id in the media handle table.
VSID_MEDIA_ID_ENTRY (int, VST_MEDIA_ID)	Index of the medium in the media handle table. Pointer to the location to store the media identifier.

Parameter Type	Description
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	Pointer to the media type supported by this MediaClass group. Valid media type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_NOTIFY_COMMENT (VST_NOTIFY_COMMENT)	Pointer to the user-specified comment to be included in a system log message when the number of media in the MediaClass group exceeds the high mark threshold or drops below the low mark threshold.
VSID_NUMBER_MEDIA_HANDLES (int *)	Pointer to the number of media handles in the media handle table.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER *)	If `VSID_CLASS_RPC_OPTION` is set to `VSE_CLASS_RPC_STANDARD`, a pointer to the RPC procedure number of the “listening task.” Otherwise, `VSID_PROCEDURE_NUMBER` is not applicable.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER *)	If `VSID_CLASS_RPC_OPTION` is set to `VSE_CLASS_RPC_STANDARD`, a pointer to the RPC program number of the client process to receive MediaClass notification messages from VolServ. If `the VSID_CLASS_RPC_OPTION` is set to `VSE_CLASS_RPC_NONE` or `VSE_CLASS_RPC_ENTERPRISE`, `VSID_PROGRAM_NUMBER` is not applicable.
VSID_PROTOCOL (VST_PROTOCOL *)	If the `VSID_CLASS_RPC_OPTION` is set to `VSE_CLASS_RPC_STANDARD`, a pointer where Internet protocol to use to return unsolicited callbacks to the client. Valid `VSID_PROTOCOL` values are enumerated in the *vs_types.h* file.
VSID_RELEASE_PRIORITY (VST_PRIORITY *)	Pointer to the release priority for this MediaClass group.

Parameter Type	Description
VSID_VERSION_NUMBER (VST_VERSION_NUMBER *)	If the VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_STANDARD, a pointer to the RPC version number of the "listening task." Otherwise, VSID_VERSION_NUMBER is not applicable.

Return Values

VS_MediaClass_GetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a MediaClass handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_mediaclass
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a media class handle.
8  *
9  * PARAMETERS:
10 * h : the media class handle to print
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
```

```
14    void
         vst_print_mediaclass(VST_MEDIACLA
         SS_HANDLEh)
15 #else
16    void vst_print_mediaclass(h)
17    VST_MEDIACLASS_HANDLE h;
18 #endif
19 {
20    VST_MEDIA_CLASS_NAME    MediaClass;
21    VST_MEDIA_TYPE_NAME
         MediaTypeName;
22    VST_PRIORITY
         ReleasePriority;
23    VST_CAPACITY            Capacity;
24    VST_FILL_LEVEL          FillLevel;
25    VST_CLASS_MOUNT_STATE   MountState;
26    VST_HIGH_MARK           HighMark;
27    VST_CLASS_RPC_OPTION    RPC_Option;
28    VST_HOSTNAME
         RPC_HostName;
29    VST_PROGRAM_NUMBER      RPC_ProgNum;
30    VST_VERSION_NUMBER      RPC_VersNum;
31    VST_PROCEDURE_NUMBER    RPC_ProcNum;
32    VST_PROTOCOL
         RPC_Protocol;
33    VST_ENTERPRISE_ID
         EnterpriseID;
34    VST_NOTIFY_COMMENT
         NotifyComment;
35    VST_TABLE_HANDLE
         MediaHandleTable;
36    int                     NumEntries;
37    int                     i;
38    VST_MEDIA_HANDLE        Media;
39
40    VS_MediaClass_GetFields(h,
41          VSID_MEDIA_CLASS_NAME,
         MediaClass,
42          VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
43          VSID_RELEASE_PRIORITY,
         &ReleasePriority,
```

```
44          VSID_CAPACITY,
          &Capacity,
45          VSID_FILL_LEVEL,
          &FillLevel,
46          VSID_CLASS_MOUNT_STATE,
          &MountState,
47          VSID_HIGH_MARK,
          &HighMark,
48          VSID_CLASS_RPC_OPTION,
          &RPC_Option,
49          VSID_HOST_NAME,
          RPC_HostName,
50          VSID_PROGRAM_NUMBER,
          &RPC_ProgNum,
51          VSID_VERSION_NUMBER,
          &RPC_VersNum,
52          VSID_PROCEDURE_NUMBER,
          &RPC_ProcNum,
53          VSID_PROTOCOL,
          &RPC_Protocol,
54          VSID_ENTERPRISE_ID,
          &EnterpriseID,
55          VSID_NOTIFY_COMMENT,
          NotifyComment,
56
          VSID_MEDIA_HANDLE_TABLE,&MediaHan
          dleTable,
57          VSID_ENDFIELD);
58    printf("****** Media Class Handle
          ******\n");
59    printf("Media Class = %s\n",
          MediaClass);
60    printf("Media Type = %s\n",
          MediaTypeName);
61    printf("Release Priority=%d\n",
          ReleasePriority);
62    printf("Capacity = %d\n", Capacity);
63    printf("Fill Level = %d\n",
          FillLevel);
64    printf("Mount State = %d\n",
          MountState);
65    printf("High Mark = %d\n", HighMark);
```

```
66    printf("RPC Option = %d\n",
         RPC_Option);
67    printf("Host Name = %s\n",
         RPC_HostName);
68    rintf("Program Number = %ld\n",
         RPC_ProgNum);
69    printf("Version Number = %d\n",
         RPC_VersNum);
70    printf("Procedure Number = %d\n",
         RPC_ProcNum);
71    printf("RPC Protocol = %d\n",
         RPC_Protocol);
72    printf("Enterprise ID = %ld\n",
         EnterpriseID);
73    printf("Notify Comment = %s\n",
         NotifyComment);
74    if ( MediaHandleTable !=
         (VST_TABLE_HANDLE) NULL)
75    {
76
         VS_Table_GetFields(MediaHandleTab
         le,
77                  VSID_NUMBER_ENTRIES,
         &NumEntries,
78                  VSID_ENDFIELD);
79       for (i = 0; i < NumEntries; i++)
80       {
81
         VS_Table_GetFields(MediaHandleTab
         le,
82                  VSID_TABLE_ENTRY, i,
         &Media,
83                  VSID_ENDFIELD);
84          vst_print_media(Media);
85       }
86    }
87 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Media_GetFields(l),
- VS_MediaClass_Create(l),
- VS_MediaClass_Destroy(l),
- VS_MediaClass_SetFields(l),
- VS_Table_Create(l),
- VS_Table_Destroy(l),
- VS_Table_GetFields(l),
- VS_Table_SetFields(l),
- VSCMD_MediaClassQuery(l)

VS_ MediaClass_ SetFields

`VS_MediaClass_SetFields` sets the value for one or more fields in a specified MediaClass handle. A MediaClass handle is used to pass MediaClass information to and from VolServ.

Synopsis

VST_BOOLEAN VS_MediaClass_SetFields
(VST_MEDIACLASS_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The MediaClass handle where the information is stored.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CAPACITY (VST_CAPACITY)	Maximum number of media allowed in this MediaClass group.
VSID_CLASS_MOUNT_STATE (VST_CLASS_MOUNT_STATE)	Indicates whether this MediaClass group supports the "mount by MediaClass" functionality. Valid values for this field are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_CLASS_RPC_OPTION (VST_CLASS_RPC_OPTION)	Indicates whether callbacks are to be activated for this MediaClass group and if they are, which callback scheme is to be used. Valid `VSID_CLASS_RPC_OPTION` values are enumerated in the *vs_types.h* file.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	If the `VSID_CLASS_RPC_OPTION` is set to `VSE_CLASS_RPC_ENTERPRISE`, the identifier of the enterprise to receive unsolicited callbacks. Otherwise, `VSID_ENTERPRISE_ID` is not applicable.
VSID_FILL_LEVEL (VST_FILL_LEVEL)	Current number of media in this MediaClass group.
VSID_HIGH_MARK (VST_HIGH_MARK)	Percentage of the MediaClass capacity above which notification or automatic media migration is initiated.
VSID_HOST_NAME (VST_HOSTNAME)	If `VSID_CLASS_RPC_OPTION` is set to `VSE_CLASS_RPC_STANDARD`, the network-assigned name of the computer where the task that “listens” for unsolicited callbacks executes. Otherwise, `VSID_HOST_NAME` is not applicable.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Name of this MediaClass group.Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces.
VSID_MEDIA_HANDLE_TABLE (VST_TABLE_HANDLE)	Media (in table format) that belong to this MediaClass group.
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	Media type supported by this MediaClass group. Valid media type names may contain up to 16 alphanumeric characters, including spaces.

Parameter Type	Description
VSID_NOTIFY_COMMENT (VST_NOTIFY_COMMENT)	User-specified comment to be included in a system log message when the number of media in the MediaClass group exceeds the high mark threshold. The MediaClass name, fill level, high mark threshold, and capacity values are automatically included in the system log message and need not be included in VSID_NOTIFY_COMMENT.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	If VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_STANDARD, the RPC procedure number of the "listening task." Otherwise, VSID_PROCEDURE_NUMBER is not applicable.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	If VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_STANDARD, the RPC program number of the client process to receive MediaClass notification messages. If VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_NONE or VSE_CLASS_RPC_ENTERPRISE, VSID_PROGRAM_NUMBER is not applicable.
VSID_PROTOCOL (VST_PROTOCOL)	If VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_STANDARD, the Internet protocol to use to return unsolicited callbacks to the client. Valid VSID_PROTOCOL values are enumerated in the *vs_types.h* file.
VSID_RELEASE_PRIORITY (VST_PRIORITY)	Release priority for this MediaClass group.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	If VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_STANDARD, the RPC version number of the "listening task." Otherwise, VSID_VERSION_NUMBER is not applicable.

Return Values

`VS_MediaClass_SetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a MediaClass handle.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediaclass_handle
4  *
5  * PURPOSE:
6  * This function tests a mediaclass
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_mediaclass_handle(void)
14 #else
15    VST_BOOLEAN vst_mediaclass_handle()
```

```
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIACLASS_HANDLE   h;
20    VST_MEDIA_CLASS_NAME    MediaClass;
21    VST_MEDIA_TYPE_NAME
         MediaTypeName;
22    VST_PRIORITY
         ReleasePriority;
23    VST_CAPACITY            Capacity;
24    VST_FILL_LEVEL          FillLevel;
25    VST_CLASS_MOUNT_STATE   MountState;
26    VST_HIGH_MARK           HighMark;
27    VST_CLASS_RPC_OPTION    RPC_Option;
28    VST_HOSTNAME
         RPC_HostName;
29    VST_PROGRAM_NUMBER      RPC_ProgNum;
30    VST_VERSION_NUMBER      RPC_VersNum;
31    VST_PROCEDURE_NUMBER    RPC_ProcNum;
32    VST_PROTOCOL
         RPC_Protocol;
33    VST_ENTERPRISE_ID
         EnterpriseID;
34    VST_NOTIFY_COMMENT
         NotifyComment;
35
36    /* create the handle */
37    h = VS_MediaClass_Create();
38    if (h != (VST_MEDIACLASS_HANDLE)
         NULL)
39    {
40       /* get values from user */
41       printf("*** Media Class Handle
         ***\n");
42       printf("Enter mediaclass name ==>
         ");
43       gets(MediaClass);
44       printf("Enter media type name ==>
         ");
45       gets(MediaTypeName);
```

```
46       printf("Enter release priority
         ==> ");
47       ReleasePriority =
         atoi(gets(input));
48       printf("Enter capacity ==> ");
49       Capacity = atoi(gets(input));
50       printf("Enter fill level ==> ");
51       FillLevel = atoi(gets(input));
52       printf("Enter mount state ==> ");
53       MountState = atoi(gets(input));
54       printf("Enter high mark ==> ");
55       HighMark = atoi(gets(input));
56       printf("Enter class RPC option ==>
         ");
57       RPC_Option = atoi(gets(input));
58       printf("Enter RPC host name ==>
         ");
59       gets(RPC_HostName);
60       printf("Enter RPC program number
         ==> ");
61       RPC_ProgNum = atol(gets(input));
62       printf("Enter RPC version number
         ==> ");
63       RPC_VersNum = atol(gets(input));
64       printf("Enter RPC procedure
         number ==> ");
65       RPC_ProcNum = atol(gets(input));
66       printf("Enter RPC protocol ==> ");
67       RPC_Protocol = atoi(gets(input));
68       printf("Enter enterprise id ==>
         ");
69       EnterpriseID = atol(gets(input));
70       printf("Enter notify comment ==>
         ");
71       gets(NotifyComment);
72       /* set the fields */
73       rc = VS_MediaClass_SetFields(h,
74          VSID_MEDIA_CLASS_NAME,
         MediaClass,
75          VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
```

```
76          VSID_RELEASE_PRIORITY,
          ReleasePriority,
77          VSID_CAPACITY,
          Capacity,
78          VSID_FILL_LEVEL,
          FillLevel,
79          VSID_CLASS_MOUNT_STATE,
          MountState,
80          VSID_HIGH_MARK,
          HighMark,
81          VSID_CLASS_RPC_OPTION,
          RPC_Option,
82          VSID_HOST_NAME,
          RPC_HostName,
83          VSID_PROGRAM_NUMBER,
          RPC_ProgNum,
84          VSID_VERSION_NUMBER,
          RPC_VersNum,
85          VSID_PROCEDURE_NUMBER,
          RPC_ProcNum,
86          VSID_PROTOCOL,
          RPC_Protocol,
87          VSID_ENTERPRISE_ID,
          EnterpriseID,
88          VSID_NOTIFY_COMMENT,
          NotifyComment,
89          VSID_ENDFIELD);
90       if (rc)
91       {
92          vst_print_mediaclass(h);
93       }
94       VS_MediaClass_Destroy(h);
95    }
96    return(rc);
97 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_MediaClass_Create(l),
- VS_MediaClass_Destroy(l),
- VS_MediaClass_GetFields(l),
- VS_Table_Create(l),
- VS_Table_Destroy(l),
- VS_Table_GetFields(l),
- VS_Table_SetFields(l)

VS_ MediaType_ Create

VS_MediaType_Create allocates a VolServ API media type handle. A media type handle is used to pass media type information to and from VolServ.

Synopsis

VST_MEDIATYPE_HANDLE VS_MediaType_Create (void)

Arguments

None

Return Values

VS_MediaType_Create returns:

- A media type handle, if one can be allocated
- NULL, if a media type handle could be allocated. An appropriate error code is set in VSG_Error.
- VSE_ERR_OUTOFMEM - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediatype_handle
4  *
5  * PURPOSE:
6  * This function tests a mediatype
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
       vst_mediatype_handle(void)
14 #else
```

```
15    VST_BOOLEAN vst_mediatype_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIATYPE_HANDLE    h;
20    VST_MEDIA_TYPE_NAME
         MediaTypeName;
21    int                     NumberSides;
22    VST_MEDIA_TYPE_CAPACITY Capacity;
23
24    h = VS_MediaType_Create();
25    if (h != (VST_MEDIATYPE_HANDLE) NULL)
26    {
27       /* get values from user */
28       printf("Enter Media Type Name ==>
         ");
29       gets(MediaTypeName);
30       printf("Enter number of sides ==>
         ");
31       NumberSides = atoi(gets(input));
32       printf("Enter media type capacity
         ==> ");
33       Capacity = atof(gets(input));
34       rc = VS_MediaType_SetFields(h,
35          VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
36          VSID_NUMBER_SIDES,
         NumberSides,
37          VSID_MEDIA_TYPE_CAPACITY,
         Capacity,
38          VSID_ENDFIELD);
39       if (rc)
40       {
41          vst_print_mediatype(h);
42       }
43       VS_MediaType_Destroy(h);
44    }
45    return(rc);
46 }
```

Notes

None

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_MediaType_Destroy(l),
- VS_MediaType_GetFields(l),
- VS_MediaType_SetFields(l)

VS_ MediaType_ Destroy

VS_MediaType_Destroy deallocates a VolServ API media type handle that was allocated with VS_MediaType_Create. A media type handle is used to pass information to and from VolServ.

Synopsis

VST_BOOLEAN VS_MediaType_Destroy (VST_MEDIATYPE_HANDLE handle)

Arguments

- handle = The media type handle to be destroyed.

Return Values

VS_MediaType_Destroy returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a media type handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediatype_handle
4  *
5  * PURPOSE:
6  * This function tests a mediatype
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_mediatype_handle(void)
14 #else
15    VST_BOOLEAN vst_mediatype_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIATYPE_HANDLE    h;
20    VST_MEDIA_TYPE_NAME
         MediaTypeName;
21    int                     NumberSides;
22    VST_MEDIA_TYPE_CAPACITY Capacity;
23
24    h = VS_MediaType_Create();
25    if (h != (VST_MEDIATYPE_HANDLE) NULL)
26    {
27       /* get values from user */
28       printf("Enter Media Type Name ==>
         ");
29       gets(MediaTypeName);
30       printf("Enter number of sides ==>
         ");
31       NumberSides = atoi(gets(input));
32       printf("Enter media type capacity
         ==> ");
33       Capacity = atof(gets(input));
34       rc = VS_MediaType_SetFields(h,
35          VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
36          VSID_NUMBER_SIDES,
         NumberSides,
37          VSID_MEDIA_TYPE_CAPACITY,
         Capacity,
38          VSID_ENDFIELD);
39       if (rc)
40       {
41          vst_print_mediatype(h);
42       }
```

```
43       VS_MediaType_Destroy(h);
44    }
45    return(rc);
46 }
```

Notes

After `VS_MediaType_Destroy` has been called for a media type handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_MediaType_Create(l),
- VS_MediaType_GetFields(l),
- VS_MediaType_SetFields(l)

VS_ MediaType_ GetFields

`VS_MediaType_GetFields` retrieves information associated with a media type handle. A media type handle is used to pass media type information to and from VolServ.

Synopsis

VST_BOOLEAN VS_MediaType_GetFields
(VST_MEDIATYPE_HANDLE handle,
"…"
VSID_ENDFIELD)

Arguments

- `handle` = The media type handle where information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` =Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_MEDIA_TYPE_CAPACITY (VST_MEDIA_TYPE_CAPACITY *)	Pointer to the capacity (in megabytes) of a medium belonging to this media type classification.
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	Pointer to the name of this media type classification. Valid media type names may contain up to 16 alphanumeric characters, including spaces.

Parameter Type	Description
VSID_NUMBER_SIDES (int *)	Pointer to the number of sides a medium belonging to this media type classification supports.

Return Values

`VS_MediaType_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a media type handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_mediatype
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a media type handle.
8  *
9  * PARAMETERS:
10 * h : the media type handle to print
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
```

```
14    void
         vst_print_mediatype(VST_MEDIATYPE
         _HANDLE h)
15 #else
16    void vst_print_mediatype(h)
17    VST_MEDIATYPE_HANDLE h;
18 #endif
19 {
20    VST_MEDIA_TYPE_NAME
         MediaTypeName;
21    int                      NumberSides;
22    VST_MEDIA_TYPE_CAPACITY Capacity;
23
24    VS_MediaType_GetFields(h,
25       VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
26       VSID_NUMBER_SIDES,
         &NumberSides,
27       VSID_MEDIA_TYPE_CAPACITY,
         &Capacity,
28       VSID_ENDFIELD);
29    printf("*** Media Type Handle
         ***\n");
30    printf("media type name = %s\n",
         MediaTypeName);
31    printf("number of sides = %d\n",
         NumberSides);
32    printf("Capacity = %.2f\n",
         Capacity);
33 }
```


Notes

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_MediaType_Create(l),
- VS_MediaType_Destroy(l),
- VS_MediaType_SetFields(l),
- VSCMD_MediaTypeQuery(l)

VS_ MediaType_ SetFields

`VS_MediaType_SetFields` sets the value for one or more fields associated with a media type handle. A media type handle is used to pass media type information to and from VolServ.

Synopsis

VST_BOOLEAN VS_MediaType_SetFields
(VST_MEDIATYPE_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The media type handle where information is stored or updated.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_MEDIA_TYPE_CAPACITY (VST_MEDIA_TYPE_CAPACITY)	Capacity (in megabytes) of a medium belonging to this media type classification.
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	Name of this media type classification. Valid media type names may contain up to 16 alphanumeric characters, including spaces. No leading or trailing spaces are permitted.

Parameter Type	Description
VSID_NUMBER_SIDES (int)	Number of sides a medium belonging to this media type classification supports.

Return Values

`VS_MediaType_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a media type handle.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediatype_handle
4  *
5  * PURPOSE:
6  * This function tests a mediatype
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_mediatype_handle(void)
14 #else
15    VST_BOOLEAN vst_mediatype_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIATYPE_HANDLE    h;
20    VST_MEDIA_TYPE_NAME
         MediaTypeName;
21    int                     NumberSides;
22    VST_MEDIA_TYPE_CAPACITY Capacity;
23
24    h = VS_MediaType_Create();
25    if (h != (VST_MEDIATYPE_HANDLE) NULL)
26    {
27       /* get values from user */
28       printf("Enter Media Type Name ==>
         ");
29       gets(MediaTypeName);
30       printf("Enter number of sides ==>
         ");
31       NumberSides = atoi(gets(input));
32       printf("Enter media type capacity
         ==> ");
33       Capacity = atof(gets(input));
34       rc = VS_MediaType_SetFields(h,
35          VSID_MEDIA_TYPE_NAME,
         MediaTypeName,
36          VSID_NUMBER_SIDES,
         NumberSides,
37          VSID_MEDIA_TYPE_CAPACITY,
         Capacity,
38          VSID_ENDFIELD);
39       if (rc)
40       {
41          vst_print_mediatype(h);
42       }
```

```
43        VS_MediaType_Destroy(h);
44     }
45     return(rc);
46  }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_MediaType_Create(l),
- VS_MediaType_Destroy(l),
- VS_MediaType_GetFields(l),
- VSCMD_MediaTypeQuery(l)

VS_Mount_Create

`VS_Mount_Create` function allocates a VolServ API mount handle. A mount handle is used to pass mount information to and from VolServ.

Synopsis

VST_MOUNT_HANDLE VS_Mount_Create (void)

Arguments

None

Return Values

`VS_Mount_Create` returns:

- A mount handle, if one can be allocated.
- NULL, if the mount handle cannot be allocated. An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_handle
4  *
5  * PURPOSE:
6  * This function creates the mount handle
       and sets the
7  * values in it according to user input.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_MOUNT_HANDLE
         vst_create_mount_handle ( void )
15 #else
```

```
16    VST_MOUNT_HANDLE
         vst_create_mount_handle ()
17 #endif
18 {
19    int                        i;
20    int                        entry;
21    VST_DRIVE_ID               driveid;
22    VST_DRIVE_POOL_NAME
         drivepool;
23    VST_MEDIA_ID               mediaid;
24    VST_MEDIA_CLASS_NAME
         mediaclass;
25    VST_MOUNT_HANDLE           mounth;
26    VST_CRITERIAGROUP_HANDLE   grouph;
27
28    /* create the handle */
29    mounth = VS_Mount_Create();
30    if ( mounth == (VST_MOUNT_HANDLE)
         NULL )
31    {
32       return ( (VST_MOUNT_HANDLE) NULL
         );
33    }
34    /* prompt user for values */
35    printf ( "Mount by (1) Media ID or (2)
         Media Class ==> " );
36    entry = atoi(gets(input));
37
38    if ( entry == 1 )
39    {
40       printf ( "Enter Media ID for
         mounting ==> " );
41       gets(mediaid);
42       VS_Mount_SetFields ( mounth,
43                VSID_MEDIA_ID, mediaid,
44                VSID_ENDFIELD );
45    }
46    else
47    {
48       printf ( "Enter Media Class for
         mounting ==> " );
49       gets(mediaclass);
```

```
50       VS_Mount_SetFields ( mounth,
51                VSID_MEDIA_CLASS_NAME,
         mediaclass,
52                VSID_ENDFIELD );
53
54       printf ( “Reclassify media (1) yes
         or (2) no ==> “ );
55       entry = atoi(gets(input));
56
57       if ( entry == 1 )
58       {
59          printf ( “Enter Target Media
         Class ==> “ );
60          gets(mediaclass);
61          VS_Mount_SetFields( mounth,
62
         VSID_TARGET_MEDIA_CLASS_NAME,
         mediaclass,
63               VSID_ENDFIELD );
64       }
65    }
66    printf ( “Mount by (1) Drive ID or (2)
         Drive Pool ==> “ );
67    entry = atoi(gets(input));
68
69    if ( entry == 1 )
70    {
71       printf ( “Enter Drive ID for
         mounting ==> “ );
72       driveid = atoi(gets(input));
73       VS_Mount_SetFields ( mounth,
74                      VSID_DRIVE_ID,
         driveid,
75                      VSID_ENDFIELD );
76    }
77    else
78    {
79       printf ( “Enter Drive Pool for
         mount ==> “ );
80       gets(drivepool);
81       VS_Mount_SetFields ( mounth,
```

```
82                   VSID_DRIVEPOOL_NAME,
      drivepool,
83                   VSID_ENDFIELD );
84    }
85    printf ( “Mount by criteria (1) yes
      or (2) no ==> “ );
86    entry = atoi(gets(input));
87
88    if ( entry == 1 )
89    {
90       printf ( “Enter number of criteria
      groups ==> “ );
91       entry = atoi(gets(input));
92
93       for ( i = 0 ; i < entry ; i++ )
94       {
95          /* create a criteria group
      handle */
96          grouph =
      vst_create_mount_criteria();
97          if ( grouph !=
      (VST_CRITERIAGROUP_HANDLE) NULL )
98          {
99             VS_Mount_SetFields (
      mounth,
100            VSID_CRITERIA_GROUP_HANDLE,
      i, grouph,
101            VSID_ENDFIELD );
102         }
103      }
104   }
105   return ( mounth );
106}
```

Notes

`VS_Mount_Create` is used with the Mount or Multimount commands.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Mount_Destroy(l),
- VS_Mount_GetFields(l),
- VS_Mount_SetFields(l),
- VSCMD_Mount(l),
- VSCMD_MultiMount(l)

VS_Mount_Destroy

`VS_Mount_Destroy` deallocates a VolServ mount handle that was allocated with `VS_Mount_Create`. A mount handle is used to pass mount information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Mount_Destroy
(VST_MOUNT_HANDLE handle)

Arguments

- handle = The mount handle to be destroyed.

Return Values

`VS_Mount_Destroy` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADHANDLE` - Specified handle was not a mount handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_multimount_execute
4  *
5  * PURPOSE:
6  * This function will test the
       VSCMD_Multimount call.
7  *
8  * PARAMETERS:
9  * none
10
11 *************************************
       *********/
```

```
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_multimount_execute(void)
14 #else
15    VST_BOOLEAN vst_multimount_execute()
16 #endif
17 {
18    int i;
19    int num;
20    VST_BOOLEAN rc = VSE_FALSE;
21    VST_COMMAND_HANDLE cmdh;
22    VST_MOUNT_HANDLE
         mounth[VSD_MAX_MOUNT_REQS];
23
24    /* get parameters from user */
25    printf("*** MultiMount Parameters
         ***\n");
26    printf("Enter the number of mount
         requests ==> " );
27    num = atoi(gets(input));
28
29    /* loop through the number of mount
         request */
30    for ( i = 0 ; i < num ; i++ )
31    {
32       /* Create a mount handle. */
33       /* Each mount handle stores a
         single mount */
34       /* request.The MultiMount command
         accepts */
35       /* multiple mount requests and
         executes */
36       /* them all as one operation.*/
37       mounth[i] =
         vst_create_mount_handle();
38       if ( mounth[i] !=
         (VST_MOUNT_HANDLE) NULL )
39       {
40          /* add the mount request to the
         multimount */
41          /* via the command default
         function */
```

```
42          VSCMD_MultiMount_SetDefaults (
43                   VSID_MOUNT_HANDLE, i,
       mounth[i],
44                   VSID_ENDFIELD );
45       }
46       else
47       {
48          rc = VSE_FALSE;
49          break;
50       }
51    }
52
53    if ( rc )
54    {
55       cmdh = VS_Command_Create();
56       if (cmdh != (VST_COMMAND_HANDLE)
       NULL)
57       {
58          /* execute the multimount
       command, note */
59          /* that all parameters have
       been set via */
60          /* default functions if sync,
       we will */
61          /* wait for all mounts to
       complete */
62          /* if async, we will leave once
       initial */
63          /* status has been returned */
64          rc = VSCMD_MultiMount ( cmdh,
65
       VSID_ENDFIELD );
66       }
67       else
68       {
69          rc = VSE_FALSE;
70       }
71    }
72
73    /* destroy the mount handles that
       contain the */
74    /* individual mount requests. */
```

```
75    for ( i = 0 ; i < num ; i++ )
76    {
77       VS_Mount_Destroy ( mounth[i] );
78    }
79    return ( rc );
80 }
```

Notes

After `VS_Mount_Destroy` has been called for a mount handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Mount_Create(l),
- VS_Mount_GetFields(l),
- VS_Mount_SetFields(l),
- VSCMD_Mount(l),
- VSCMD_MultiMount(l)

VS_ MediaClass_ GetFields

VS_Mount_GetFields retrieves information associated with a mount handle. A mount handle is used to pass mount information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Mount_GetFields
(VST_MOUNT_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- handle = The mount handle where information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CRITERIA_GROUP_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the table of criteria group handles associated with the mount handle.
VSID_DRIVE_SELECT (VST_DRIVE_SELECT *)	Pointer to the type of drive selection (drive identifier or drive pool) to associate with the mount command. Valid VSID_DRIVE_SELECT values are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_DRIVE_ID (VST_DRIVE_ID *)	Pointer to the drive identifier to mount when mounting by drive identifier.
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	Pointer to the name of a drive pool group from which to select a drive when mounting by drive pool group. Valid DrivePool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_DRIVE_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the list of drives to exclude from the given drive pool group when mounting by drive pool group.
VSID_LOCK_ID (VST_LOCK_ID *)	Pointer to the lock identifier that is required if a drive is locked.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Pointer to the MediaClass name from which a medium is selected to mount. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MEDIA_ID (VST_MEDIA_ID)	Pointer to the identifier of the medium to be mounted.
VSID_MEDIA_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the media identifier list (in table format) from which the medium to be mounted is selected when mounting by media list.
VSID_MEDIA_SELECT (VST_MEDIA_SELECT *)	Pointer to the type of media selection (media identifier, media list, or MediaClass group).
VSID_MOUNT_OPTION (VST_MOUNT_OPTION *)	Pointer to a flag that indicates which mount processing options are in effect for the mount command. Valid `VSID_MOUNT_OPTION` values are listed in the *vs_defs.h* file.

Parameter Type	Description
VSID_MOVEWAIT_OPTION (VST_MOVEWAIT_OPTION *)	Pointer to a flag that indicates how VolServ is to process a mount request that requires an inter-archive move. Valid VSID_MOVEWAIT_OPTION values are enumerated in the *vs_types.h* file.
VSID_TARGET_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Pointer to the MediaClass name where the mounted media is reclassified if the reclassify option is in effect for the mount command.

Return Values

VS_Mount_GetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a mount handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_mount
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a mount handle.
8  * PARAMETERS:
9  * mounth : the mount handle to print
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    void
         vst_print_mount(VST_MOUNT_HANDLE
         mounth)
14 #else
15    void vst_print_mount(mounth)
16    VST_MOUNT_HANDLE mounth;
17 #endif
18 {
19    int                          i, size;
20    VST_DRIVE_SELECT
         driveselect;
21    VST_DRIVE_ID                 driveid;
22    VST_DRIVE_POOL_NAME
         drivepool;
23    VST_MEDIA_SELECT
         mediaselect;
24    VST_MEDIA_ID                 mediaid;
25    VST_MEDIA_CLASS_NAME
         mediaclass;
26    VST_MEDIA_CLASS_NAME
         targetmediaclass;
27    VST_MOUNT_OPTION             mountopt;
28    VST_CRITERIAGROUP_HANDLE     grouph;
29    VST_TABLE_HANDLE             tableh;
30
31    /* check for a null handle */
32    if ( mounth == (VST_MOUNT_HANDLE)
         NULL )
33    {
34       printf("error...mount handle is
         null\n");
35       return;
36    }
37
38    VS_Mount_GetFields ( mounth,
39                VSID_DRIVE_SELECT,
         &driveselect,
40                VSID_MEDIA_SELECT,
         &mediaselect,
```

```
41                VSID_MOUNT_OPTION,
      &mountopt,
42                VSID_ENDFIELD );
43
44    printf("**** Mount Handle Values
      ****\n");
45    switch ( driveselect )
46    {
47       case VSE_DRIVE_SELECT_ID:
48          VS_Mount_GetFields ( mounth,
49                VSID_DRIVE_ID,
      &driveid,
50                VSID_ENDFIELD );
51          printf ( "Drive ID: %d\n",
      driveid );
52          break;
53       case VSE_DRIVE_SELECT_POOL:
54          VS_Mount_GetFields ( mounth,
55                VSID_DRIVEPOOL_NAME,
      drivepool,
56                VSID_ENDFIELD );
57          printf ( "Drive Pool: %s\n",
      drivepool );
58          break;
59       default:
60          printf ( "error...incorrect
      drive select value\n");
61          break;
62    }
63
64    switch ( mediaselect )
65    {
66       case VSE_MEDIA_SELECT_ID:
67          VS_Mount_GetFields ( mounth,
68                VSID_MEDIA_ID,
      mediaid,
69                VSID_ENDFIELD );
70             printf ( "Media ID: %s\n",
      mediaid );
71             break;
72       case VSE_MEDIA_SELECT_LIST:
73          printf ( "Media List:\n" );
```

```
74
75          VS_Mount_GetFields ( mounth,
76                VSID_MEDIA_ID_TABLE,
          &tableh,
77                VSID_ENDFIELD );
78
79          VS_Table_GetFields ( tableh,
80                VSID_NUMBER_ENTRIES,
          &size,
81                VSID_ENDFIELD );
82
83          for ( i = 0 ; i < size ; i++ )
84          {
85             VS_Table_GetFields (
          tableh,
86                VSID_TABLE_ENTRY, i,
          &mediaid,
87                VSID_ENDFIELD );
88
89             printf ( "%s", mediaid );
90          }
91          break;
92       case VSE_MEDIA_SELECT_CLASS:
93          VS_Mount_GetFields(mounth,
94                VSID_MEDIA_CLASS_NAME,
          mediaclass,
95                VSID_ENDFIELD);
96          printf( "Media Class: %s\n",
          mediaclass);
97
98          if (mountopt &
          VSD_MOUNT_OPTION_RECLASS)
99          {
100            VS_Mount_GetFields(mounth,
101
          VSID_TARGET_MEDIA_CLASS_NAME,
          targetmediaclass,
102               VSID_ENDFIELD );
103            printf("Target Media Class:
          %s\n",
                  targetmediaclass );
104         }
```

```
105         break;
106      default:
107         printf("error ...incorrect
         media select
               value\n");
108         break;
109   }
110
111   if (mountopt &
         VSD_MOUNT_OPTION_CRITERIA)
112   {
113      VS_Mount_GetFields(mounth,
114
         VSID_CRITERIA_GROUP_HANDLE_TABLE,
         &tableh,
115         VSID_ENDFIELD);
116
117      VS_Table_GetFields(tableh,
118            VSID_NUMBER_ENTRIES,
         &size,
119            VSID_ENDFIELD);
120
121      for (i = 0; i < size; i++)
122      {
123         VS_Table_GetFields(tableh,
124            VSID_TABLE_ENTRY, i,
         &grouph,
125            VSID_ENDFIELD);
126
127
         vst_print_criteria_group(grouph);
128      }
129   }
130   return;
131}
```

Notes

The mount handle contains all relevant mount information. The user can set all mount parameters in a mount handle and pass the mount handle to the Mount or Multimount command. The Mount or Multimount command retrieves the required information from the handle.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Mount_Create(l),
- VS_Mount_Destroy(l),
- VS_Mount_SetFields(l),
- VS_Table_GetFields(l),
- VSCMD_Mount(l),
- VSCMD_MultiMount(l)

VS_Mount_SetFields

VS_Mount_SetFields sets the value for one or more fields associated with the mount handle. A mount handle is used to pass mount information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Mount_SetFields
(VST_MOUNT_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- handle = The mount handle where information is stored.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CRITERIA_GROUP_HANDLE (int)	Number of the specified criteria group. A criteria group number can be 0 through 3, inclusive.
(VST_CRITERIAGROUP_HANDLE)	A criteria group to be used by the mount command in selecting media for mounting.
VSID_DRIVE_ID (VST_DRIVE_ID)	Drive identifier to mount when mounting by drive identifier. If VSID_DRIVE_ID is specified, VSID_DRIVEPOOL_NAME and VSID_DRIVE_EXCL_LIST cannot be specified.

Parameter Type	Description
VSID_DRIVEPOOL_NAME (VS_DRIVE_POOL_NAME)	Drive pool group from which a drive is selected to mount when mounting by drive pool group. If `VSID_DRIVEPOOL_NAME` is specified, `VSID_DRIVE_ID` cannot be specified. Valid DrivePool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_DRIVE_EXCL_LIST (int)	Number of drives to exclude from the specified drive pool group when mounting by drive pool group.
(VST_DRIVE_ID *)	List of drives to exclude from the specified drive pool group when mounting by drive pool group.
VSID_LOCK_ID (VST_LOCK_ID)	Lock identifier that is required if a drive is locked.
VSID_MEDIA_ID (VST_MEDIA_ID)	Identifier of the medium to be mounted when mounting by medium identifier. If `VSID_MEDIA_ID` is specified, `VSID_MEDIA_CLASS_NAME` and `VSID_MEDIA_ID_LIST` cannot be specified.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Name of the MediaClass group from which a medium is selected to mount when mounting by MediaClass group. If `VSID_MEDIA_CLASS_NAME` is specified, `VSID_MEDIA_ID` and `VSID_MEDIA_ID_LIST` cannot be specified.
VSID_MEDIA_ID_LIST (int)	Number of media in the list.
(`char *`)	List of media from which one medium is selected for mounting. If `VSID_MEDIA_ID_LIST` is specified, `VSID_MEDIA_CLASS_NAME` and `VSID_MEDIA_ID` cannot be specified.

Parameter Type	Description
VSID_MOUNT_OPTION (VST_MOUNT_OPTION)	Flag that indicates which mount processing options are in effect for the mount command. Valid VSID_MOUNT_OPTION values are listed in the *vs_defs.h* file.
VSID_MOVEWAIT_OPTION (VST_MOVEWAIT_OPTION)	Flag that indicates how VolServ is to process a mount request that requires an inter-archive move. Valid VSID_MOVEWAIT_OPTION values are enumerated in the *vs_types.h* file.
VSID_TARGET_MEDIA_CLASS_NAME (VST_TARGET_MEDIA_CLASS_NAME)	Name of the MediaClass group where the mounted medium is reclassified if the reclassify option is in effect for the mount command.

Return Values

VS_Mount_SetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a mount handle.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

- VSE_ERR_OUTOFRANGE - Specified entry does not exist in the table's range of values.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_create_mount_handle
4  *
5  * PURPOSE:
6  * This function creates the mount handle
       and sets the
7  * values in it according to user input.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_MOUNT_HANDLE
         vst_create_mount_handle ( void )
15 #else
16    VST_MOUNT_HANDLE
         vst_create_mount_handle ()
17 #endif
18 {
19    int                        i;
20    int                        entry;
21    VST_DRIVE_ID               driveid;
22    VST_DRIVE_POOL_NAME
         drivepool;
23    VST_MEDIA_ID               mediaid;
24    VST_MEDIA_CLASS_NAME
         mediaclass;
25    VST_MOUNT_HANDLE           mounth;
26    VST_CRITERIAGROUP_HANDLE   grouph;
27
28    /* create the handle */
29    mounth = VS_Mount_Create();
30    if ( mounth == (VST_MOUNT_HANDLE)
         NULL )
```

```
31    {
32       return ( (VST_MOUNT_HANDLE) NULL
         );
33    }
34    /* prompt user for values */
35    printf ( “Mount by (1) Media ID or (2)
         Media Class ==> “ );
36    entry = atoi(gets(input));
37
38    if ( entry == 1 )
39    {
40       printf ( “Enter Media ID for
         mounting ==> “ );
41       gets(mediaid);
42       VS_Mount_SetFields ( mounth,
43             VSID_MEDIA_ID, mediaid,
44             VSID_ENDFIELD );
45    }
46    else
47    {
48       printf ( “Enter Media Class for
         mounting ==> “ );
49       gets(mediaclass);
50       VS_Mount_SetFields ( mounth,
51             VSID_MEDIA_CLASS_NAME,
         mediaclass,
52             VSID_ENDFIELD );
53
54       printf ( “Reclassify media (1) yes
         or (2) no ==> “ );
55       entry = atoi(gets(input));
56
57       if ( entry == 1 )
58       {
59          printf ( “Enter Target Media
         Class ==> “ );
60          gets(mediaclass);
61          VS_Mount_SetFields( mounth,
62
         VSID_TARGET_MEDIA_CLASS_NAME,
         mediaclass,
63             VSID_ENDFIELD );
```

```
64          }
65       }
66       printf ( “Mount by (1) Drive ID or (2)
            Drive Pool ==> “ );
67       entry = atoi(gets(input));
68
69       if ( entry == 1 )
70       {
71          printf ( “Enter Drive ID for
            mounting ==> “ );
72          driveid = atoi(gets(input));
73          VS_Mount_SetFields ( mounth,
74                         VSID_DRIVE_ID,
            driveid,
75                         VSID_ENDFIELD );
76       }
77       else
78       {
79          printf ( “Enter Drive Pool for
            mount ==> “ );
80          gets(drivepool);
81          VS_Mount_SetFields ( mounth,
82
            VSID_DRIVEPOOL_NAME, drivepool,
83                         VSID_ENDFIELD );
84       }
85       printf ( “Mount by criteria (1) yes
            or (2) no ==> “ );
86       entry = atoi(gets(input));
87
88       if ( entry == 1 )
89       {
90          printf ( “Enter number of criteria
            groups ==> “ );
91          entry = atoi(gets(input));
92
93          for ( i = 0 ; i < entry ; i++ )
94          {
95             /* create a criteria group
            handle */
96             grouph =
            vst_create_mount_criteria();
```

```
97          if ( grouph !=
           (VST_CRITERIAGROUP_HANDLE) NULL )
98          {
99             VS_Mount_SetFields (
           mounth,
100
           VSID_CRITERIA_GROUP_HANDLE, i,
           grouph,
101                VSID_ENDFIELD );
102         }
103      }
104   }
105   return ( mounth );
106}
```

Notes

The mount handle contains all relevant mount information. The user can set all mount parameters in a mount handle and pass the mount handle to the Mount or Multimount command. The Mount or Multimount command retrieves the required information from the handle.

The `VSID_CRITERIA_GROUP_HANDLE`, `VSID_DRIVE_EXCL_LIST`, and `VSID_MEDIA_ID_LIST` parameters require that two arguments be passed instead of one. The first argument passed is the entry number in the appropriate table. The second argument is a pointer to the values for setting.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Mount_Create(l),
- VS_Mount_Destroy(l),
- VS_Mount_GetFields(l),
- VSCMD_Mount(l),
- VSCMD_MultiMount(l)

VS_Notify_Create

`VS_Notify_Create` allocates a VolServ API notify handle. A notify handle is used by the API to allow a client to listen for MediaClass notifications.

Synopsis

VST_NOTIFY_HANDLE VS_Notify_Create (void)

Arguments

None

Return Values

`VS_Notify_Create` returns:

- A notify handle, if one can be allocated
- NULL, if a notify handle could not be allocated. An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_OUTOFMEM` - Memory allocation call failed.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_notify
4  *
5  * PURPOSE:
6  * This routine is used to test the notify
       loop.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_notify( void )
14 #else
15    VST_BOOLEAN
16    vst_notify()
```

```
17 #endif
18 {
19    VST_BOOLEAN          done, rc;
20    int                  i, count,
         timeout;
21    unsigned long        prognum,
         versnum, procnum;
22    VST_TIME_OUT         t;
23    VST_TABLE_HANDLE     table;
24    VST_MEDIA_CLASS_NAME class,
         newclass;
25    VST_NOTIFY_HANDLE    h;
26
27    rc = VSE_TRUE;
28    done = VSE_FALSE;
29    NumNotifies = 0;
30
31    /* get parameters from user */
32    printf("*** Notify Parameters ***\n"
         );
33    printf("Program Number ==> " );
34    prognum = (VST_PROGRAM_NUMBER)
         atol(gets(input));
35
36    printf("Version Number ==> " );
37    versnum = (VST_VERSION_NUMBER)
         atol(gets(input));
38
39    printf("Procedure Number ==> " );
40    procnum = (VST_PROCEDURE_NUMBER)
         atol(gets(input));
41
42    printf("Number of Notifies to listen
         ==> " );
43    count = atoi(gets(input));
44
45    printf("Timeout Value ==> " );
46    t = atoi(gets(input));
47
48    printf("Number of Timeouts to process
         ==> " );
49    timeout = atoi(gets(input));
```

```
50
51    printf(“\nlistening...\n” );
52
53    /* create the notify handle */
54    if ( (h = VS_Notify_Create()) !=
         (VST_NOTIFY_HANDLE) NULL )
55    {
56       /* initialize the notify handle */
57       VS_Notify_SetFields ( h,
58          VSID_PROTOCOL, VSE_PROT_TCP,
59          VSID_PROGRAM_NUMBER, prognum,
60          VSID_VERSION_NUMBER, versnum,
61          VSID_PROCEDURE_NUMBER,procnum,
62          VSID_CLIENT_DISPATCH,
         vst_notify_dispatch,
63          VSID_TIMEOUT_VALUE,   t,
64          VSID_ENDFIELD );
65
66       done = VSE_FALSE;
67       while ( ! done )
68       {
69          /* “listen” for callbacks and
         act on the */
70          /* error code */
71          switch ( (i = VS_Notify_Listen(
         h )) )
72          {
73             case VSE_ERR_TIMEOUT:
74                printf(“Timed out\n” );
75                timeout--;
76                break;
77             case VSE_ERR_NONE:
78                /* This is the successful
         case. */
79                /* Nothing is printed
         here because */
80                /* the notify handle is
         printed in */
81                /* vst_notify_dispatch
         */
82                break;
83             default:
```

```
84                printf("Select Error
        %d\n", i );
85                rc = VSE_FALSE;
86                done = VSE_TRUE;
87                break;
88             }
89
90             if ( NumNotifies > count )
91             {
92                printf("Number of Notifies
        reached\n" );
93                done = VSE_TRUE;
94             }
95
96             if ( timeout <= 0 )
97             {
98                printf("Timeout value
        reached\n" );
99                done = VSE_TRUE;
100            }
101         }
102
103         /* destroy the notify handle */
104         VS_Notify_Destroy ( h );
105      }
106      else
107      {
108         rc = VSE_FALSE;
109      }
110
111      return ( rc );
112   }
```


Notes None

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Notify_Destroy(l),
- VS_Notify_GetFields(l),
- VS_Notify_Listen(l),
- VS_Notify_SetFields(l)

VS_Notify_Destroy

VS_Notify_Destroy deallocates a VolServ API notify handle that was allocated with VS_Notify_Create.

Synopsis

VST_BOOLEAN VS_Notify_Destroy
(VST_NOTIFY_HANDLE notifyhandle)

Arguments

- notifyhandle = The notify handle to be destroyed.

Return Values

VS_Notify_Destroy returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a notify handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_notify
4  *
5  * PURPOSE:
6  * This routine is used to test the notify
       loop.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
```

```
13    VST_BOOLEAN vst_notify( void )
14 #else
15    VST_BOOLEAN
16    vst_notify()
17 #endif
18 {
19    VST_BOOLEAN          done, rc;
20    int                  i, count,
         timeout;
21    unsigned long        prognum,
         versnum, procnum;
22    VST_TIME_OUT         t;
23    VST_TABLE_HANDLE     table;
24    VST_MEDIA_CLASS_NAME class,
         newclass;
25    VST_NOTIFY_HANDLE    h;
26
27    rc = VSE_TRUE;
28    done = VSE_FALSE;
29    NumNotifies = 0;
30
31    /* get parameters from user */
32    printf("*** Notify Parameters ***\n"
         );
33    printf("Program Number ==> " );
34    prognum = (VST_PROGRAM_NUMBER)
         atol(gets(input));
35
36    printf("Version Number ==> " );
37    versnum = (VST_VERSION_NUMBER)
         atol(gets(input));
38
39    printf("Procedure Number ==> " );
40    procnum = (VST_PROCEDURE_NUMBER)
         atol(gets(input));
41
42    printf("Number of Notifies to listen
         ==> " );
43    count = atoi(gets(input));
44
45    printf("Timeout Value ==> " );
46    t = atoi(gets(input));
```

```
47
48    printf("Number of Timeouts to process
         ==> " );
49    timeout = atoi(gets(input));
50
51    printf("\nlistening...\n" );
52
53    /* create the notify handle */
54    if ( (h = VS_Notify_Create()) !=
         (VST_NOTIFY_HANDLE) NULL )
55    {
56       /* initialize the notify handle */
57       VS_Notify_SetFields ( h,
58          VSID_PROTOCOL, VSE_PROT_TCP,
59          VSID_PROGRAM_NUMBER,
         prognum,
60          VSID_VERSION_NUMBER,
         versnum,
61          VSID_PROCEDURE_NUMBER,
         procnum,
62          VSID_CLIENT_DISPATCH,
         vst_notify_dispatch,
63          VSID_TIMEOUT_VALUE,     t,
64          VSID_ENDFIELD );
65
66       done = VSE_FALSE;
67       while ( ! done )
68       {
69          /* "listen" for callbacks and
         act on the */
70          /* error code */
71          switch ( (i = VS_Notify_Listen(
         h )) )
72          {
73             case VSE_ERR_TIMEOUT:
74                printf("Timed out\n" );
75                timeout--;
76                break;
77             case VSE_ERR_NONE:
78                /* This is the successful
         case. */
```

```
79                /* Nothing is printed
        here because */
80                /* the notify handle is
        printed in */
81                /* vst_notify_dispatch
        */
82                break;
83             default:
84                printf("Select Error
        %d\n", i );
85                rc = VSE_FALSE;
86                done = VSE_TRUE;
87                break;
88          }
89
90          if ( NumNotifies > count )
91          {
92             printf("Number of Notifies
        reached\n" );
93             done = VSE_TRUE;
94          }
95
96          if ( timeout <= 0 )
97          {
98             printf("Timeout value
        reached\n" );
99             done = VSE_TRUE;
100         }
101      }
102
103      /* destroy the notify handle */
104      VS_Notify_Destroy ( h );
105   }
106   else
107   {
108      rc = VSE_FALSE;
109   }
110
111   return ( rc );
112}
```

Notes

After `VS_Notify_Destroy` has been called for a notify handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Notify_Create(l),
- VS_Notify_GetFields(l),
- VS_Notify_Listen(l),
- VS_Notify_SetFields(l)

VS_Notify_GetFields

`VS_Notify_GetFields` retrieves information associated with a notify handle. A notify handle is used by the API to allow a client to listen for MediaClass notifications.

Synopsis

VST_BOOLEAN VS_Notify_GetFields
(VST_NOTIFY_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The notify handle where information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Pointer to the archive name for this callback. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_DRIVE_ID (VST_DRIVEID *)	Pointer to the drive identifier for this callback.
VSID_DRIVE_ID_ENTRY (int)	Index of a specific entry in the drive identifier table.

Parameter Type	Description
(VST_DRIVE_ID *)	Pointer to a specific entry in the drive identifier table.
VSID_DRIVE_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the drive identifiers (in table format) associated with this callback.
VSID_ERROR_HANDLE (VST_ERROR_HANDLE *)	Pointer to the error handle for this callback.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Pointer to the MediaClass name for this callback.
VSID_MEDIA_ID (VST_MEDIA_ID)	Pointer to the media identifier for this callback.
VSID_MEDIA_ID_ENTRY (int)	Number of media in the media identifier table.
(VST_MEDIA_ID *)	Pointer to the media identifier table.
VSID_MEDIA_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the media identifiers (in table format) associated with this callback.
VSID_NOTIFY_TYPE (VST_NOTIFY_TYPE *)	Pointer to the type of VolServ command that generated this callback. Valid `VSID_NOTIFY_TYPE` values are enumerated in the *vs_types.h* file.
VSID_NUMBER_DRIVE_IDS (int *)	Pointer to the number of drive ids present in the drive id table.
VSID_NUMBER_MEDIA_IDS (int *)	Pointer to the number of media ids present in the media id table.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER *)	Pointer to the RPC procedure number of the client process to receive callbacks generated for this MediaClass group.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER *)	Pointer to the RPC program number of the client process to receive MediaClass callbacks.

Parameter Type	Description
VSID_PROTOCOL (VST_PROTOCOL *)	Pointer to the RPC protocol the API should use for callbacks. Valid `VSID_PROTOCOL` values for this field are enumerated in the *vs_types.h* file.
VSID_STATUS_CODE (VST_STATUS_CODE *)	Pointer to the status code stating whether the enter or eject passed or failed. `VSID_STATUS_CODE` is returned only for enter and eject callbacks.
VSID_TARGET_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Pointer to the target MediaClass name for this callback.
VSID_TIMEOUT_VALUE (VST_TIME_OUT *)	Pointer to the amount of time (in seconds) the API software is to wait for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER *)	Pointer to the RPC version number the API should use to receive callbacks.

Return Values

`VS_Notify_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD`- An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a notify handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_notify_dispatch
4  *
5  * PURPOSE:
6  * This is the dispatch routine used to
       test the
7  * notify loop.
8  *
9  * PARAMETERS:
10 * h : the notify handle to print
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    int      vst_notify_dispatch (
       VST_NOTIFY_HANDLE h )
15 #else
16    int      vst_notify_dispatch ( h )
17    VST_NOTIFY_HANDLE h;
18 #endif
19 {
20    int                     i, n;
21    VST_ARCHIVE_NAME        archive;
22    VST_DRIVE_ID            * did;
23    char                    * mid;
24    VST_MEDIA_CLASS_NAME    class,
       newclass;
25    VST_NOTIFY_TYPE         type;
26    VST_ERROR_HANDLE        eh;
27    VST_TABLE_HANDLE        dt, mt;
28    VST_STATUS_CODE         sc;
29
30    /* increment a counter for the number
       of */
31    /* notifies processed */
32    NumNotifies++;
33    /* initialize values in the notify
       handle. */
34    VS_Notify_GetFields ( h,
```

```
35            VSID_NOTIFY_TYPE,
         &type,
36            VSID_ARCHIVE_NAME,
         archive,
37            VSID_MEDIA_CLASS_NAME,
         class,
38
         VSID_TARGET_MEDIA_CLASS_NAME,
         newclass,
39            VSID_DRIVE_ID_TABLE,
         &dt,
40            VSID_MEDIA_ID_TABLE,
         &mt,
41            VSID_ERROR_HANDLE,
         &eh,
42            VSID_ENDFIELD );
43
44    printf("***************** Notify
         Handle Information
         *************\\pn" );
45    printf("Notify Type = %d\n", type );
46    if ((type == VSE_NOTIFY_EJECT) ||
         (type == VSE_NOTIFY_ENTER))
47    {
48       VS_Notify_GetFields ( h,
49
         VSID_STATUS_CODE, &sc,
50
         VSID_ENDFIELD );
51       if (sc == VSE_STATUS_OK)
52          printf("Successful\n");
53       else
54          printf("Failure\n");
55    }
56    printf("Archive Name = %s\n", archive
         );
57    printf("Media Class = %s\n", class );
58    switch(type)
59    {
60       case VSE_NOTIFY_IMPORT:
61       case VSE_NOTIFY_EXPORT:
62       case VSE_NOTIFY_ENTER:
```

```
63       case VSE_NOTIFY_EJECT:
64       case VSE_NOTIFY_CHECKIN:
65       case VSE_NOTIFY_CHECKOUT:
66       case VSE_NOTIFY_RECLASSIFY:
67          /* media callbacks */
68          /*  -- print the media table */
69          if (type ==
         VSE_NOTIFY_RECLASSIFY)
70          {
71             printf("New Media Class =
         %s\n", newclass);
72          }
73          if ( mt != (VST_TABLE_HANDLE)
         NULL )
74          {
75             VS_Table_GetFields ( mt,
76
         VSID_NUMBER_ENTRIES, &n,
77                          VSID_ENDFIELD );
78             for ( i = 0 ; i < n ; i++ )
79             {
80                VS_Table_GetFields ( mt,
81
         VSID_TABLE_ENTRY, i, &mid,
82                           VSID_ENDFIELD
         );
83                printf("Media ID Entry #
         %d = %s\n",
         i, mid );
84             }
85          }
86          break;
87       case VSE_NOTIFY_MOUNT:
88       case VSE_NOTIFY_DISMOUNT:
89          /* print the drive and media id
         */
90          VS_Notify_GetFields(h,
91
         VSID_MEDIA_ID, mid,
92
         VSID_DRIVE_ID, &did,
```

```
93
          VSID_ENDFIELD);
94           printf("media id = %s\n", mid);
95           printf("drive id = %d\n", did);
96           break;
97        default:
98           break;
99     }
100    /* print the error handle */
101    vst_print_error ( eh );
102}
```

Notes

The `VSID_DRIVE_ID_ENTRY` and `VSID_MEDIA_ID_ENTRY` parameters require that two arguments be retrieved instead of one. The first argument retrieved is the entry number in the appropriate table. The second argument is a pointer to the location where the value should be stored.

After the client receives a MediaClass notification from `VS_Notify_Listen`, the Notify handle contains the notify information. To determine the Notify type, the client then retrieves the relevant information from the Notify handle (i.e., for a Mount notification, the media identifier and drive identifier must be retrieved).

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Notify_Create(l),
- VS_Notify_Destroy(l),
- VS_Notify_Listen (l),
- VS_Notify_SetFields(l),
- VS_Table_GetFields(l)

VS_Notify_Listen

VS_Notify_Listen listens for MediaClass callbacks from VolServ.

Client registers a notify dispatch routine with the API using either the VS_Global_SetFields or the VS_Notify_SetFields function. Client is then responsible for putting the API into a "listening" state by calling VS_Notify_Listen. VS_Notify_Listen terminates either after receiving a MediaClass callback and notifying the client's notify dispatch routine or after timing out.

Synopsis

int VS_Notify_Listen (VST_NOTIFY_HANDLE handle)

Arguments

- handle = The notify handle used to listen for MediaClass callbacks.

Return Values

VS_Notify_Listen returns:

- VSE_ERR_BADHANDLE - Specified handle was not a notify handle.
- VSE_ERR_NONE - Successfully received and processed a callback.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLHANDLE - Specified handle was a null handle.
- VSE_ERR_SELECT - An error occurred during a select system call.
- VSE_ERR_SIGNAL - A select system call was interrupted by a signal.

- VSE_ERR_TIMEOUT - The VolServ API timed out waiting for MediaClass callbacks from VolServ.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_notify
4  *
5  * PURPOSE:
6  * This routine is used to test the notify
       loop.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_notify( void )
14 #else
15    VST_BOOLEAN
16    vst_notify()
17 #endif
18 {
19    VST_BOOLEAN          done, rc;
20    int                  i, count,
         timeout;
21    unsigned long        prognum,
         versnum, procnum;
22    VST_TIME_OUT         t;
23    VST_TABLE_HANDLE     table;
24    VST_MEDIA_CLASS_NAME class,
         newclass;
25    VST_NOTIFY_HANDLE    h;
26
27    rc = VSE_TRUE;
28    done = VSE_FALSE;
29    NumNotifies = 0;
30
31    /* get parameters from user */
```

```
32    printf("*** Notify Parameters ***\n"
         );
33    printf("Program Number ==> " );
34    prognum = (VST_PROGRAM_NUMBER)
         atol(gets(input));
35
36    printf("Version Number ==> " );
37    versnum = (VST_VERSION_NUMBER)
         atol(gets(input));
38
39    printf("Procedure Number ==> " );
40    procnum = (VST_PROCEDURE_NUMBER)
         atol(gets(input));
41
42    printf("Number of Notifies to listen
         ==> " );
43    count = atoi(gets(input));
44
45    printf("Timeout Value ==> " );
46    t = atoi(gets(input));
47
48    printf("Number of Timeouts to process
         ==> " );
49    timeout = atoi(gets(input));
50
51    printf("\nlistening...\n" );
52
53    /* create the notify handle */
54    if ( (h = VS_Notify_Create()) !=
         (VST_NOTIFY_HANDLE) NULL )
55    {
56       /* initialize the notify handle */
57       VS_Notify_SetFields ( h,
58          VSID_PROTOCOL, VSE_PROT_TCP,
59          VSID_PROGRAM_NUMBER,
         prognum,
60          VSID_VERSION_NUMBER, versnum,
61          VSID_PROCEDURE_NUMBER,
         procnum,
62          VSID_CLIENT_DISPATCH,
         vst_notify_dispatch,
63          VSID_TIMEOUT_VALUE,     t,
```

```
64          VSID_ENDFIELD );
65
66       done = VSE_FALSE;
67       while ( ! done )
68       {
69          /* “listen” for callbacks and
         act on the */
70          /* error code */
71          switch ( (i = VS_Notify_Listen(
         h )) )
72          {
73             case VSE_ERR_TIMEOUT:
74                printf(“Timed out\n” );
75                timeout--;
76                break;
77             case VSE_ERR_NONE:
78                /* This is the successful
         case. */
79                /* Nothing is printed
         here because */
80                /* the notify handle is
         printed in */
81                /* vst_notify_dispatch
         */
82                break;
83             default:
84                printf(“Select Error
         %d\n”, i );
85                rc = VSE_FALSE;
86                done = VSE_TRUE;
87                break;
88          }
89
90          if ( NumNotifies > count )
91          {
92             printf(“Number of Notifies
         reached\n” );
93             done = VSE_TRUE;
94          }
95
96          if ( timeout <= 0 )
97          {
```

```
98             printf("Timeout value
          reached\n" );
99             done = VSE_TRUE;
100         }
101      }
102
103      /* destroy the notify handle */
104      VS_Notify_Destroy ( h );
105   }
106   else
107   {
108      rc = VSE_FALSE;
109   }
110
111   return ( rc );
112}
```

Notes

Client uses `VS_Notify_GetFields` to access the MediaClass callback information in a notify handle.

If a `VSE_ERR_SIGNAL` is returned, any client error handling routines that have been installed for that signal have been called.

`VS_Notify_Listen` returns error codes instead of the normal `VST_BOOLEAN` values (`VSE_TRUE, VSE_FALSE`). This simplifies client code when performing asynchronous processing. Client can use the "switch" statement on the return code directly from the routine without having to retrieve error codes.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Notify_Create(l),
- VS_Notify_Destroy(l),
- VS_Notify_GetFields(l),
- VS_Notify_SetFields(l)

VS_Notify_SetFields

`VS_Notify_SetFields` sets the value of one or more fields in a notify handle. A notify handle is used to pass notify information to and from VolServ.

Synopsis

```
VST_BOOLEAN VS_Notify_SetFields
(VST_NOTIFY_HANDLE handle,
"…",
VSID_ENDFIELD)
```

Arguments

- `handle` = The notify handle where information is stored or updated.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_NOTIFY_DISPATCH (VST_NOTIFY_DISPATCH)	Function in the client's program that the VolServ API calls when a MediaClass callback is received.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	RPC procedure number of the client process that the VolServ API calls when a MediaClass callback is received.

Parameter Type	Description
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	RPC program number of the client process that the VolServ API calls when a MediaClass callback is received. If VSID_PROGRAM_NUMBER is not specified, the program number defaults to 0. When VSID_PROGRAM_NUMBER is allowed to default to 0, the API software obtains a transient RPC number.
VSID_PROTOCOL (VST_PROTOCOL)	RPC protocol that the VolServ API uses for callbacks. The default VSID_PROTOCOL is VSE_PROT_TCP.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) that the VolServ API waits for a MediaClass callback from VolServ before timing out.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	RPC version number of the client process that the VolServ API calls when a MediaClass callback is received.

Return Values

VS_Notify_SetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a notify handle.
- VSE_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_notify
4  *
5  * PURPOSE:
6  * This routine is used to test the notify
       loop.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_notify( void )
14 #else
15    VST_BOOLEAN
16    vst_notify()
17 #endif
18 {
19    VST_BOOLEAN          done, rc;
20    int                  i, count,
         timeout;
21    unsigned long        prognum,
         versnum, procnum;
22    VST_TIME_OUT         t;
23    VST_TABLE_HANDLE     table;
24    VST_MEDIA_CLASS_NAME class,
         newclass;
25    VST_NOTIFY_HANDLE    h;
26
27    rc = VSE_TRUE;
28    done = VSE_FALSE;
29    NumNotifies = 0;
30
31    /* get parameters from user */
```

```
32    printf("*** Notify Parameters ***\n"
         );
33    printf("Program Number ==> " );
34    prognum = (VST_PROGRAM_NUMBER)
         atol(gets(input));
35
36    printf("Version Number ==> " );
37    versnum = (VST_VERSION_NUMBER)
         atol(gets(input));
38
39    printf("Procedure Number ==> " );
40    procnum = (VST_PROCEDURE_NUMBER)
         atol(gets(input));
41
42    printf("Number of Notifies to listen
         ==> " );
43    count = atoi(gets(input));
44
45    printf("Timeout Value ==> " );
46    t = atoi(gets(input));
47
48    printf("Number of Timeouts to process
         ==> " );
49    timeout = atoi(gets(input));
50
51    printf("\nlistening...\n" );
52
53    /* create the notify handle */
54    if ( (h = VS_Notify_Create()) !=
         (VST_NOTIFY_HANDLE) NULL )
55    {
56       /* initialize the notify handle */
57       VS_Notify_SetFields ( h,
58          VSID_PROTOCOL, VSE_PROT_TCP,
59          VSID_PROGRAM_NUMBER,
         prognum,
60          VSID_VERSION_NUMBER,
         versnum,
61          VSID_PROCEDURE_NUMBER,
         procnum,
62          VSID_CLIENT_DISPATCH,
         vst_notify_dispatch,
```

```
63          VSID_TIMEOUT_VALUE,     t,
64          VSID_ENDFIELD );
65
66       done = VSE_FALSE;
67       while ( ! done )
68       {
69          /* “listen” for callbacks and
         act on the */
70          /* error code */
71          switch ( (i = VS_Notify_Listen(
         h )) )
72          {
73             case VSE_ERR_TIMEOUT:
74                printf(“Timed out\n” );
75                timeout--;
76                break;
77             case VSE_ERR_NONE:
78                /* This is the successful
         case. */
79                /* Nothing is printed
         here because */
80                /* the notify handle is
         printed in */
81                /* vst_notify_dispatch
         */
82                break;
83             default:
84                printf(“Select Error
         %d\n”, i );
85                rc = VSE_FALSE;
86                done = VSE_TRUE;
87                break;
88          }
89
90          if ( NumNotifies > count )
91          {
92             printf(“Number of Notifies
         reached\n” );
93             done = VSE_TRUE;
94          }
95
96          if ( timeout <= 0 )
```

```
97          {
98              printf("Timeout value
          reached\n" );
99              done = VSE_TRUE;
100         }
101      }
102
103      /* destroy the notify handle */
104      VS_Notify_Destroy ( h );
105   }
106   else
107   {
108      rc = VSE_FALSE;
109   }
110
111   return ( rc );
112}
```


Notes

Client dispatch function must be prototypes as follows:

void NotifyClientDispatch (`VST_NOTIFY_HANDLE` handle)

RPC information (protocol, program, procedure, and version) must be set to match the callback information for the MediaClass group the user wants to monitor.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VS_Notify_Create(l),
- VS_Notify_Destroy(l),
- VS_Notify_GetFields(l),
- VS_Notify_Listen(l)

VS_Request_Create

VS_Request_Create allocates a VolServ API request handle. A request handle is used to pass request information to and from VolServ.

Synopsis

VST_REQUEST_HANDLE VS_Request_Create (void)

Arguments

None

Return Values

VS_Request_Create returns:

- A request handle, if one can be allocated.
- NULL, if a request handle could not be allocated An appropriate error code is set in VSG_Error.
- VSE_ERR_OUTOFMEM - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_request_handle
4  *
5  * PURPOSE:
6  * This function tests a request handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_request_handle(void)
14 #else
15    VST_BOOLEAN vst_request_handle()
16 #endif
17 {
```

```
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_REQUEST_HANDLE   h;
20    VST_REQUEST_ID       RequestID;
21    VST_REQUEST_TYPE     RequestType;
22    VST_REQUEST_STATE    State;
23    VST_PRIORITY         Priority;
24
25    /* create the handle */
26    h = VS_Request_Create();
27    if (h != (VST_REQUEST_HANDLE) NULL)
28    {
29       /* get values from user */
30       printf("*** Request Handle
         ***\n");
31       printf("Enter request id ==> ");
32       RequestID = atol(gets(input));
33       printf("Enter request type ==> ");
34       RequestType = atoi(gets(input));
35       printf("Enter request state ==>
         ");
36       State = atoi(gets(input));
37       printf("Enter Priority ==> ");
38       Priority = atoi(gets(input));
39       /* set the fields */
40       rc = VS_Request_SetFields(h,
41                   VSID_REQUEST_ID,
         RequestID,
42                   VSID_REQUEST_TYPE,
         RequestType,
43                   VSID_REQUEST_STATE,
         State,
44                   VSID_PRIORITY,
         Priority,
45                   VSID_ENDFIELD);
46       if (rc)
47       {
48          vst_print_request(h);
49       }
50       VS_Request_Destroy(h);
51    }
52    return(rc);
53 }
```

Notes

None

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Request_Destroy(l),
- VS_Request_GetFields(l),
- VS_Request_SetFields(l)

VS_Request_Destroy

VS_Request_Destroy deallocates a VolServ API request handle that was allocated with VS_Request_Create. A request handle is used to pass request information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Request_Destroy
(VST_REQUEST_HANDLE handle)

Arguments

- handle = The request handle to be destroyed.

Return Values

VS_Request_Destroy returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a request handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_request_handle
4  *
5  * PURPOSE:
6  * This function tests a request handle.
7  *
8  * PARAMETERS:
9  * none
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_request_handle(void)
14 #else
15    VST_BOOLEAN vst_request_handle()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_REQUEST_HANDLE   h;
20    VST_REQUEST_ID       RequestID;
21    VST_REQUEST_TYPE     RequestType;
22    VST_REQUEST_STATE    State;
23    VST_PRIORITY         Priority;
24
25    /* create the handle */
26    h = VS_Request_Create();
27    if (h != (VST_REQUEST_HANDLE) NULL)
28    {
29       /* get values from user */
30       printf("*** Request Handle
         ***\n");
31       printf("Enter request id ==> ");
32       RequestID = atol(gets(input));
33       printf("Enter request type ==> ");
34       RequestType = atoi(gets(input));
35       printf("Enter request state ==>
         ");
36       State = atoi(gets(input));
37       printf("Enter Priority ==> ");
38       Priority = atoi(gets(input));
39       /* set the fields */
40       rc = VS_Request_SetFields(h,
41                   VSID_REQUEST_ID,
         RequestID,
42                   VSID_REQUEST_TYPE,
         RequestType,
43                   VSID_REQUEST_STATE,
         State,
44                   VSID_PRIORITY,
         Priority,
45                   VSID_ENDFIELD);
```

```
46       if (rc)
47       {
48          vst_print_request(h);
49       }
50       VS_Request_Destroy(h);
51    }
52    return(rc);
53 }
```

Notes

After `VS_Request_Destroy` has been called for a request handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Request_Create(l),
- VS_Request_GetFields(l),
- VS_Request_SetFields(l)

VS_Request_GetFields

`VS_Request_GetFields` retrieves information associated with a request handle. A request handle is used to pass request information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Request_GetFields
(VST_REQUEST_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The request handle where information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_PRIORITY (VST_PRIORITY *)	Pointer to the execution priority of this request.
VSID_REQUEST_ID (VST_REQUEST_ID *)	Pointer to the request identifier associated with the indicated request handle.
VSID_REQUEST_STATE (VST_REQUEST_STATE *)	Pointer to the execution state of the request. Valid `VSID_REQUEST_STATE` values are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_REQUEST_TYPE (VST_REQUEST_TYPE *)	Pointer to the request type of the request. Valid VSID_REQUEST_TYPE values are enumerated in the *vs_types.h* file.
VSID_TIME (VST_TIME *)	Pointer to the arrival time of the request.

Return Values

VS_Request_GetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a request handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_request
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a request handle.
8  *
9  * PARAMETERS:
10 * h : the request handle to print
11 *
12 *************************************
       *********/
```

```
#ifdef ANSI_C
   void
      vst_print_request(VST_REQUEST_HAN
      DLE h)
#else
   void vst_print_request(h)
   VST_REQUEST_HANDLE h;
#endif
{
   VST_REQUEST_ID       RequestID;
   VST_REQUEST_TYPE     RequestType;
   VST_TIME             InTime;
   VST_REQUEST_STATE    State;
   VST_PRIORITY         Priority;

   VS_Request_GetFields(h,
               VSID_REQUEST_ID,
      &RequestID,
               VSID_REQUEST_TYPE,
      &RequestType,
               VSID_TIME,
      &InTime,
               VSID_REQUEST_STATE,
      &State,
               VSID_PRIORITY,
      &Priority,
               VSID_ENDFIELD);
   printf("******Request
      Handle******\n");
   printf("Request ID = %d\n",
      RequestID);
   printf("Request Type = %d\n",
      RequestType);
   printf("Time = %s", ctime((time_t *)
      &InTime));
   printf("Request state = %d\n",
      State);
   printf("Priority = %d\n", Priority);
}
```

Notes

`VSID_TIME` is kept as a long integer. Use the **ctime** function to convert it to a string.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Request_Create(l),
- VS_Request_Destroy(l),
- VS_Request_SetFields(l),
- VSCMD_RequestQuery(l)

VS_Request_SetFields

`VS_Request_SetFields` sets the value of one or more fields in a request handle. A request handle is used to pass request information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Request_SetFields
(VST_REQUEST_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The request handle where information is stored or updated.
- "…" = Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_REQUEST_ID (VST_REQUEST_ID)	Identifier of the request (assigned by VolServ) associated with the specified request handle. A valid request identifier must be specified in the `yyyyddmm` format where `yyyy` represents the year, `dd` represents the day, and `mm` is a month.
VSID_REQUEST_STATE (VST_REQUEST_STATE)	The execution state of the request. Valid `VSID_REQUEST_STATE` values are enumerated in the *vs_types.h* file.
VSID_REQUEST_TYPE (VST_RQUEST_TYPE)	The request type to be associated with the request handle. Valid `VSID_REQUEST_TYPE` values are enumerated in the *vs_types.h* file.
VSID_TIME (VST_TIME)	Arrival time of the request.

Return Values

`VS_Request_SetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_FIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a request handle.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_request_handle
4  *
5  * PURPOSE:
6  * This function tests a request handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_request_handle(void)
14 #else
15    VST_BOOLEAN vst_request_handle()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_REQUEST_HANDLE   h;
20    VST_REQUEST_ID       RequestID;
21    VST_REQUEST_TYPE     RequestType;
22    VST_REQUEST_STATE    State;
23    VST_PRIORITY         Priority;
24
25    /* create the handle */
26    h = VS_Request_Create();
27    if (h != (VST_REQUEST_HANDLE) NULL)
28    {
29       /* get values from user */
30       printf("*** Request Handle
         ***\n");
31       printf("Enter request id ==> ");
32       RequestID = atol(gets(input));
33       printf("Enter request type ==> ");
34       RequestType = atoi(gets(input));
35       printf("Enter request state ==>
         ");
36       State = atoi(gets(input));
37       printf("Enter Priority ==> ");
38       Priority = atoi(gets(input));
```

```
39       /* set the fields */
40       rc = VS_Request_SetFields(h,
41                   VSID_REQUEST_ID,
         RequestID,
42                   VSID_REQUEST_TYPE,
         RequestType,
43                   VSID_REQUEST_STATE,
         State,
44                   VSID_PRIORITY,
         Priority,
45                   VSID_ENDFIELD);
46       if (rc)
47       {
48          vst_print_request(h);
49       }
50       VS_Request_Destroy(h);
51    }
52    return(rc);
53 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Request_Create(l),
- S_Request_Destroy(l),
- VS_Request_GetFields(l)

VS_Select

`VS_Select` requests the VolServ API to wait for command status from VolServ when the API in operating in asynchronous mode.

In asynchronous mode, the client issues a VolServ command through the API, immediately receives initial status for the command, issues `VS_Select` to request that the API wait for intermediate and/or final status, and continues processing. When the API receives intermediate or final status from VolServ, it calls the specified client dispatch routine to process the status and the outstanding `VS_Select` function terminates. It is the client's responsibility to issue the `VS_Select` for each intermediate and final status expected for a command.

Synopsis

int VS_Select (void)

Arguments

None

Return Values

`VS_Select` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_NONE` - Successfully received and processed status.
- `VSE_ERR_NOTINITIALIZED` - The VolServ API is not initialized.
- `VSE_ERR_SELECT` - An error occurred during a select system call.

- VSE_ERR_SIGNAL - A select system call was interrupted by a signal.
- VSE_ERR_TIMEOUT - The VolServ API timed out waiting for status from VolServ.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_select
4  *
5  * PURPOSE:
6  * This function tests the VS_Select
       loop.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_select ( void )
14 #else
15    VST_BOOLEAN vst_select ()
16 #endif
17 {
18    VST_BOOLEAN    rc = VSE_FALSE;
19    int            errcode;
20
21    errcode = VS_Select();
22    switch (errcode)
23    {
24       case VSE_ERR_NONE:
25          printf("*** No Error --
       Successful Operation ***\n");
26          rc = VSE_TRUE;
27          break;
28       case VSE_ERR_TIMEOUT:
29          printf("*** Select Timed Out
       ***\n");
30          break;
```

```
31       default:
32          printf("*** A Select Error
         Occurred ***\n");
33          break;
34    }
35    return(rc);
36 }
```

Notes

`VS_Select` calls fail if `VS_Initialize` has not been invoked.

`VS_Select` is used only when the API is operating in asynchronous mode. Therefore, clients that use synchronous processing do not use the `VS_Select` routine.

The VolServ API uses the client dispatch routine identification information, set with the `VS_Global_SetDefaults` function or with the command default function, to determine which routine to notify when intermediate and final command status is received.

The total amount time the API waits for status is `VSID_TIMEOUT_VALUE`.

`VS_Select` returns error codes instead of the normal `VST_BOOLEAN` values (`VSE_TRUE, VSE_FALSE`). This simplifies client code when performing asynchronous processing. Client can use the "switch" statement on the return code directly from the routine without having to retrieve error codes.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetDefaults(l),
- VS_Initialize(l)

VS_Status_GetFields

`VS_Status_GetFields` retrieves information associated with a status handle. A status handle is used to pass status information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Status_GetFields
(VST_STATUS_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The status handle where information is retrieved.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ACTION_CODE (VST_ACTION_CODE *)	Pointer to the first entry in the action code table.
VSID_ACTION_CODE_ENTRY (int)	Index of the appropriate entry in the action code table.
(VST_ACTION_CODE *)	Pointer to the appropriate entry in the action code table.

Parameter Type	Description
VSID_ACTION_CODE_TABLE (VST_TABLE_HANDLE *)	Pointer to the action code table associated with this status.
VSID_ARCHIVE_HANDLE (VST_ARCHIVE_HANDLE *)	Pointer to the first archive handle in the archive handle table.
VSID_ARCHIVE_HANDLE_ENTRY (int)	Index of the appropriate archive handle in the archive handle table.
(VST_ARCHIVE_HANDLE *)	Pointer to the appropriate archive handle in the archive handle table.
VSID_ARCHIVE_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the archive handle table associated with this status.
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Pointer to the name of the archive associated with this status. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ARCHIVEMEDIACLASS_HANDLE (VST_ARCHIVE_MEDIACLAS_HANDLE *)	Pointer to the first archive media class handle in the archive media class table.
VSID_ARCHIVEMEDIACLASS_HANDLE_ ENTRY (int)	Index of the appropriate archive media class handle in the archive media class handle table.
(VST_ARCHIVEMEDIACLASS_HANDLE *)	Pointer to the appropriate archive media class handle in the archive media class handle table.
VSID_ARCHIVEMEDIACLASS_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the archive media class handle table associated with this status.
VSID_COMPONENT_HANDLE (VST_COMPONENT_HANDLE *)	Pointer to the first component handle in the component handle table.
VSID_COMPONENT_HANDLE_ENTRY (int)	Index of the appropriate component handle in the component handle table.
(VST_COMPONENT_HANDLE *)	Pointer to the appropriate component handle in the component handle table.

Parameter Type	Description
VSID_COMPONENT_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the component handle table associated with this status.
VSID_COMP_ID (VST_COMPONENT_ID *)	Pointer to the first component identifier in the component identifier table.
VSID_COMP_ID_ENTRY (int)	Index of the appropriate component identifier in the component identifier table.
(VST_COMPONENT_ID *)	Pointer to the appropriate component identifier in the component identifier table.
VSID_COMP_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the component identifier table associated with this status.
VSID_COMP_STATE (VST_COMPONENT_STATE *)	Pointer to the component state for this status. Valid `VSID_COMP_STATE` values are enumerated in the `vs_types.h` file.
VSID_CONNECT_HANDLE (VST_CONNECT_HANDLE *)	Pointer to the first entry in the connect handle table.
VSID_CONNECT_HANDLE_ENTRY (int)	`I`ndex of the appropriate connect handle in the connect handle table.
(VST_CONNECT_HANDLE *)	Pointer to the appropriate connect handle in the connect handle table.
VSID_CONNECT_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the connect handle table associated with this status.
VSID_DRIVE_HANDLE (VST_DRIVE_HANDLE *)	Pointer to the first drive handle in the drive handle table.
VSID_DRIVE_HANDLE_ENTRY (int)	`I`ndex the appropriate drive handle in the drive handle table.
(VST_DRIVE_HANDLE *)	Pointer to the appropriate drive handle in the drive handle table.
VSID_DRIVE_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the drive handle table associated with this status.

Parameter Type	Description
VSID_DRIVE_ID (VST_DRIVE_ID *)	Pointer to the first drive identifier in the drive identifier table.
VSID_DRIVE_ID_ENTRY (int)	Index of the appropriate drive identifier in the drive identifier table.
(VST_DRIVE_ID *)	Pointer to the appropriate drive identifier in the drive identifier table.
VSID_DRIVE_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the drive identifier table associated with this status.
VSID_DRIVEPOOL_HANDLE (VST_DRIVE_POOL_HANDLE *)	Pointer to the first drive pool handle in the drive pool handle table.
VSID_DRIVEPOOL_HANDLE_ENTRY (int)	Index of the appropriate drive pool handle in the drive pool handle table.
(VST_DRIVEPOOL_HANDLE *)	Pointer to the appropriate drive pool handle in the drive pool handle table.
VSID_DRIVEPOOL_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the drive pool handle table associated with this status.
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	Pointer to the name of the drive pool group.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID *)	Pointer to the identifier of the enterprise, if any, to receive command status.
VSID_ERROR_CODE (VST_VOLERR_CODE *)	Pointer to the first error code in the error code table.
VSID_ERROR_CODE_ENTRY (int)	Index of the appropriate error code in the error code table.
(VST_VOLERR_CODE *)	Pointer to the appropriate error code in the error code table.
VSID_ERROR_TABLE (VST_TABLE_HANDLE *)	Pointer to the error code table associated with this status.
VSID_FIELD (int *)	Pointer to the first field in the user-defined media statistics field table.

Parameter Type	Description
VSID_FIELD_ENTRY (int)	The index of the appropriate field in the user-defined media statistics field table.
(int *)	Pointer to the appropriate field in the user-defined media statistics field table.
VSID_FIELD_TABLE (VST_TABLE_HANDLE *)	Pointer to the user-defined media statistics field table associated with this status.
VSID_LOCK_ID (VST_LOCK_ID *)	Pointer to the lock identifier associated with this status.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Pointer to the MediaClass name associated with this status.
VSID_MEDIA_HANDLE (VST_MEDIA_HANDLE *)	Pointer to the first media handle in the media handle table.
VSID_MEDIA_HANDLE_ENTRY (int)	The index of the appropriate media handle in the media handle table.
(VST_MEDIA_HANDLE *)	Pointer to the appropriate media handle in the media handle table.
VSID_MEDIA_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the media handle table associated with this status.
VSID_MEDIA_ID (VST_MEDIA_ID)	Pointer to the first media identifier in the media identifier table.
VSID_MEDIA_ID_ENTRY (int)	The index of the appropriate entry in the media identifier table.
(VST_MEDIA_ID *)	Pointer to the appropriate media identifier in the media identifier table.
VSID_MEDIA_ID_TABLE (VST_TABLE_HANDLE *)	Pointer to the media identifier table associated with this status.
VSID_MEDIACLASS_HANDLE (VST_MEDIACLASS_HANDLE *)	Pointer to the first MediaClass handle in the MediaClass handle table.
VSID_MEDIACLASS_HANDLE_ENTRY (int)	The index of the appropriate MediaClass handle in the MediaClass handle table

Parameter Type	Description
(VST_MEDIACLASS_HANDLE *)	Pointer to the appropriate MediaClass handle in the MediaClass handle table.
VSID_MEDIACLASS_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the MediaClass handle table associated with this status.
VSID_MEDIATYPE_HANDLE (VST_MEDIATYPE_HANDLE *)	Pointer to the first media type handle in the media type handle table.
VSID_MEDIATYPE_HANDLE_ENTRY (int)	The index of the appropriate media type handle in the media type handle table.
(VST_MEDIATYPE_HANDLE *)	Pointer to the appropriate media type handle in the media type handle table.
VSID_MEDIATYPE_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the media type handle table associated with this status.
VSID_NUMBER_ACTION_CODES (int *)	Pointer to the number of action codes in the action code table.
VSID_NUMBER_ARCHIVE_HANDLES (int *)	Pointer to the number of archive handles in the archive handle table.
VSID_NUMBER_COMP_IDS (int *)	Pointer to the number of component ids in the component identifier table.
VSID_NUMBER_COMPONENT_HANDLES (int *)	Pointer to the number of component handles in the component handle table.
VSID_NUMBER_CONNECT_HANDLES (int *)	Pointer to the number of connect handles in the connect handle table.
VSID_NUMBER_DRIVE_HANDLES (int *)	Pointer to the number of drive handles in the drive handle table.
VSID_NUMBER_DRIVE_IDS (int *)	Pointer to the number of drive ids in the drive id table.
VSID_NUMBER_DRIVEPOOL_HANDLES (int *)	Pointer to the number of drive pool handles in the drive pool handle table.
VSID_NUMBER_ERROR_CODES (int *)	Pointer to the number of error codes in the error code table.

Parameter Type	Description
VSID_NUMBER_FIELDS (int *)	Pointer to the number of field ids in the field identifier table.
VSID_NUMBER_MEDIA_HANDLES (int *)	Pointer to the number of media handles present in the media handle table.
VSID_NUMBER_MEDIA_IDS (int *)	Pointer to the number of media ids in the media id table.
VSID_NUMBER_MEDIACLASS_HANDLES (int *)	Pointer to the number of media class handles in the media class handle table.
VSID_NUMBER_MEDIATYPE_HANDLES (int *)	Pointer to the number of media type handles in the media type handle table.
VSID_NUMBER_REQUEST_IDS (int *)	Pointer to the number of request ids present in the request id table.
VSID_NUMBER_REQUEST_HANDLES (int *)	Pointer to the number of request handles in the request handle table.
VSID_PID (VST_PID *)	Pointer to the VolServ identifier for the Ping status.
VSID_QRY_ENTERPRISE_ID (VST_ENTERPRISE_ID *)	Pointer to the enterprise identifier, if any, for the Connect Query command.
VSID_QRY_OPTION (VST_QRY_OPTION *)	Pointer to the query option for this status.
VSID_REQUEST_HANDLE (VST_REQUEST_HANDLE *)	Pointer to the first request handle in the request handle table.
VSID_REQUEST_HANDLE_ENTRY (int)	The index of the appropriate request handle in the request handle table.
(VST_REQUEST_HANDLE *)	Pointer to the appropriate request handle in the request handle table.
VSID_REQUEST_HANDLE_TABLE (VST_TABLE_HANDLE *)	Pointer to the request handle table associated with this status.
VSID_REQUEST_ID (VST_REQUEST_ID *)	Pointer to the request identifier of the target command for a Cancel or Reprioritize request.

Parameter Type	Description
VSID_SEQUENCE_NUM (int *)	Pointer to the sequence number of this status. Initial status for a command request is sequence number 0. Sequence numbers for subsequent statuses for the same command request are assigned as one-up numbers. For example, the first intermediate status (if there is an intermediate status) or the final status (if there is no intermediate status) is sequence number 1.
VSID_SEQUENCE_TABLE (VST_TABLE_HANDLE *)	Pointer to the sequence numbers (in table format) of the statuses received for this command.
VSID_STATUS_CODE (VST_STATUS_CODE *)	Pointer to the status code for this status. Indicates whether the command was successful or failed. Valid `VSID_STATUS_CODE` values are enumerated in the *vs_types.h* file.
VSID_STATUS_TYPE (VST_STATUS_TYPE *)	Pointer to the status type (intermediate or final) for this status. Valid `VSID_STATUS_TYPE` values are enumerated in the *vs_types.h* file.
VSID_TARGET_ENTERPRISE_ID (VST_ENTERPRISE_ID *)	Pointer to the enterprise identifier for a ConnectQuery or Disconnect command.
VSID_USER_FIELD (VST_USER_FIELD)	Pointer to the user field contents for the associated command. `USER_FIELD` is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for each command. Neither the API software nor VolServ uses `USER_FIELD`.
VSID_WAIT_REASON (VST_WAIT_REASON *)	Pointer to the wait reason for an intermediate status. Valid `VSID_WAIT_REASON` values are enumerated in the *vs_types.h* file.

Return Values

`VS_Status_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a request handle.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediaquery_execute_sync
4  *
5  * PURPOSE:
6  * This executes the VSCMD_MediaQuery API
       call in a
7  * synchronous mode. It shows how to
       process status
8  * in synchronous mode without using a
       dispatch
9  * routine.
10 *
11 * PARAMETERS:
12 * none
13 *
14 *************************************
       *********/
```

```
15 #ifdef ANSI_C
16    VST_BOOLEAN
         vst_mediaquery_execute_sync(void)
17 #else
18    VST_BOOLEAN
         vst_mediaquery_execute_sync()
19 #endif
20 {
21    VST_BOOLEAN          rc = VSE_FALSE;
22    VST_QUERY_OPTION     queryopt;
23    int                  i;
24    int                  count;
25    char                 *
         medialist[VST_MAX_ITEMS];
26    VST_COMMAND_HANDLE   cmd;
27    VST_STATUS_HANDLE    status;
28    VST_TABLE_HANDLE
         mediahandletable;
29    VST_MEDIA_HANDLE     media;
30
31    /* get parameters from user */
32    printf("*** Media Query parameters
         ***\n" );
33    printf("0) Query by media list, 1)
         Query all ==> " );
34    queryopt = atoi(gets(input));
35
36    if (queryopt == 0)
37    {
38       count =
         vst_getmedialist(medialist,
         VST_MAX_ITEMS);
39    }
40
41    /* create the command handle */
42    /* Note that the command handle is
         not */
43    /* destroyed in this routine, but in
         */
44    /* vst_dispatch when final status is
         */
45    /* received */
```

```
46    cmd = VS_Command_Create();
47    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
48    {
49       /* Send the command to the VolServ
         software. */
50       /* Note that status is not
         processed here. */
51       /* Instead,it is processed in the
         */
52       /* vst_dispatch routine. */
53       /* Also, note that default values,
         such as */
54       /* timeout, value retry limit, and
         priority */
55       /* are set as default parameters.
         */
56       rc = VSCMD_MediaQuery(cmd,
57          VSID_QRY_OPTION, queryopt,
58          VSID_MEDIA_ID_LIST, count,
         medialist,
59          VSID_CLIENT_DISPATCH, NULL,
60          /* no dispatch routine */
61          VSID_STATUS_WAIT_FLAG,
         VSE_TRUE,
62          /* synchronous mode */
63          VSID_ENDFIELD);
64       if (rc)
65       {
66          /* the command was successful
         */
67          /* get the status */
68          VS_Command_GetFields(cmd,
69
         VSID_STATUS_HANDLE, &status,
70                       VSID_ENDFIELD);
71
72          /* get the media handle table
         from the */
73          /* status handle */
74          VS_Status_GetFields(status,
```

```
75             VSID_MEDIA_HANDLE_TABLE
       &mediahandletable,
76             VSID_ENDFIELD);
77
78          /* Get the number of entries in
       the table */
79
       VS_Table_GetFields(mediahandletab
       le,
80
       VSID_NUMBER_ENTRIES, &count
81                         VSID_ENDFIELD);
82
83          /* loop through the table and
       print out */
       /* the entries */
85          for (i = 0; i < count; i++)
86          {
87
       VS_Table_GetFields(mediahandletab
       le,
88                         VSID_TABLE_ENTRY,
       i, &media,
89                         VSID_ENDFIELD);
90             vst_print_media(media);
91          }
92       }
93       /* destroy the command */
94       VS_Command_Destroy(cmd);
95    }
96    return ( rc );
97 }
```

Notes

The parameters listed below require that two arguments be passed instead of one.

- The first argument passed is the index of the entry in the appropriate table.

- The second argument is a pointer to the location where the retrieved value is stored.
  - VSID_ACTION_CODE_ENTRY,
  - VSID_ARCHIVE_HANDLE_ENTRY,
  - VSID_ARCHIVEMEDIACLASS_ENTRY,
  - VSID_COMPONENT_ENTRY,
  - VSID_COMP_ID_ENTRY,
  - VSID_CONNECT_HANDLE_ENTRY,
  - VSID_DRIVE_HANDLE_ENTRY,
  - VSID_DRIVEPOOL_HANDLE_ENTRY,
  - VSID_ERROR_CODE_ENTRY,
  - VSID_FIELD_ENTRY,
  - VSID_MEDIA_HANDLE_ENTRY,
  - VSID_MEDIA_ID_ENTRY,
  - VSID_MEDIACLASS_HANDLE_ENTRY,
  - VSID_MEDIATYPE_HANDLE_ENTRY,
  - VSID_REQUEST_HANDLE_ENTRY

Table entries are zero-based (like arrays in C). The first entry in an empty table is stored in position 0, the second entry in a table is stored in position 1, and the nth entry in a table is stored in position n-1.

The status handle is associated with a command handle. To retrieve status information, the status handle must first be retrieved from the command handle using `VS_Command_GetFields`.

The `VSID_STATUS_TYPE, VSID_STATUS_CODE, VSID_USER_FIELD` and `VSID_REQUEST_ID` parameters are valid status handle fields for all VolServ requests.

The status fields that are valid for each command are identified in the "Notes" para`graph` for that command. A matrix showing which status fields are valid for which commands is in *Appendix* A.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_GetFields(l),
- VS_Error_GetFields(l)

VS_Table_AddEntry

`VS_Table_AddEntry` adds an entry to the specified table at the first available location.

When a client adds an entry to a table using `VS_Table_AddEntry`, the client is responsible for allocating and maintaining the space to contain the information maintained in the table.

Synopsis

```
VST_BOOLEAN VS_Table_AddEntry
(VST_TABLE_HANDLE handle,
void * entry)
```

Arguments

- `handle` = Handle of the table where an entry is added.
- `entry` = Pointer to the data to be stored in the table.

Return Values

`VS_Table_AddEntry` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADENTRY` - Pointer to store was null.
- `VSE_ERR_BADHANDLE` - Specified handle was not a table handle.
- `VSE_ERR_FULL` - The table is full and the entry could not be added.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_createarchivemediaclass_execu
       te
4  *
5  * PURPOSE:
6  * This executes the
       VSCMD_CreateArchiveMediaClass
7  * API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
       vst_createarchivemediaclass_execu
       te(void)
15 #else
16    VST_BOOLEAN
       vst_createarchivemediaclass_execu
       te()
17 #endif
18 {
19    int                          i;
20    int                          count;
21    VST_BOOLEAN                  rc =
       VSE_FALSE;
22    VST_ARCHIVE_NAME             archive;
23    VST_MEDIA_CLASS_NAME
       mediaclass;
24    VST_CAPACITY                 capacity;
25    VST_ARCHIVE_ACTION_OPTION    action;
26    VST_HIGH_MARK                highmark;
27    VST_LOW_MARK                 lowmark;
28    VST_PRIORITY                 migpri;
29    VST_ARCHIVE_NAME
       targetarchive;
```

```
30    VST_TABLE_HANDLE
         comphandletable;
31    VST_COMPONENT_HANDLE
         comphandle;
32    VST_COMP_TYPE  CompType =
         VSE_COMPTYPE_COLUMN;
33    ST_COMPONENT_ID            CompID;
34    VST_COMMAND_HANDLE         cmd;
35
36    bzero ( CompID, sizeof (
         VST_COMPONENT_ID ) );
37    /* get parameters from user */
38    printf("*** Create Archive Media
         Class parameters ***\n" );
39    printf("Enter Archive Name ==> " );
40    gets( archive );
41    printf("Enter Media Class Name ==> "
         );
42    gets( mediaclass );
43    printf("Enter Capacity Percent ==> "
         );
44    capacity = atoi(gets(input));
45    printf("Enter Archive action option
         (0-none/1-mig/2-notify) ==> " );
46    action = atoi(gets(input));
47    printf("Enter High Mark Percentage
         ==> " );
48    highmark = atoi(gets(input));
49    printf("Enter Low Mark Percentage ==>
         " );
50    lowmark = atoi(gets(input));
51
52    if ( action == VSE_ARCHIVE_ACTION_MIG
         )
53    {
54       /* these parameters only need to
         be set */
55       /* if the archivemediaclass is
         being */
56       /* set up to support migration */
57       printf("Enter Target Archive ==> "
         );
```

```
58          gets( targetarchive );
59          printf("Enter Migration Priority
            == > " );
60          migpri = atoi(gets(input));
61
            VSCMD_CreateArchiveMediaClass_Set
            Defaults (
62             VSID_TARGET_ARCHIVE_NAME,
            targetarchive,
63             VSID_MIGRATION_PRIORITY,
            migpri,
64             VSID_ENDFIELD );
65       }
66
67       printf("How many prefered placements
            (0 to skip): ");
68       count = atoi(gets(input));
69       if (count > 0)
70       {
71          comphandletable =
            VS_Table_Create(VSE_COMPONENT_HAN
            DLE, count);
72          if (comphandletable
            ==(VST_TABLE_HANDLE)NULL)
73          {
74             return(VSE_FALSE);
75          }
76          for (i = 0; i < count; i++)
77          {
78             printf("Enter row #%d:", i +
            1);
79             CompID[0] = (short)
            atoi(gets(input));
80             printf("Enter column #%d:", i +
            1);
81             CompID[1] = (short)
            atoi(gets(input));
82             CompID[2] = 0;
83             CompID[3] = 0;
84             comphandle =
            VS_Component_Create();
```

```
85
        VS_Component_SetFields(comphandle
        ,
86                      VSID_COMP_TYPE,
        CompType,
87                      VSID_COMP_ID,
        CompID,
88                      VSID_ENDFIELD);
89
        VS_Table_AddEntry(comphandletable
        ,comphandle);
90      }
91      /* This also only needs to be set
        if it is */
92      /* actually being used. */
93      /* It is not needed otherwise. */
94
        VSCMD_CreateArchiveMediaClass_Set
        Defaults(
95         VSID_COMPONENT_HANDLE_TABLE
        comphandletable,
96         VSID_ENDFIELD);
97    }
98
99    /* create the command handle */
100   /* Note that the command handle is not
         destroyed */
101   /* in this routine, but in
         vst_dispatch */
102   /* when final status is received. */
103   cmd = VS_Command_Create();
104   if (cmd != (VST_COMMAND_HANDLE )NULL)
105   {
106      /* Send the command to the VolServ
         software. */
107      /* Note that status is not
         processed here. */
108      /* Instead, it is processed in the
         vst_dispatch */
109      /* Also, note that default values
         such as */
```

```
110      /* timeout value retry limit, and
         priority */
111      /* are set as default parameters.
         */
112      rc =
         VSCMD_CreateArchiveMediaClass(cmd
         ,
113               VSID_ARCHIVE_NAME,
         archive,
114               VSID_MEDIA_CLASS_NAME,
         mediaclass,
115               VSID_HIGH_MARK,
         highmark,
116               VSID_LOW_MARK,
         lowmark,
117               VSID_CAPACITY,
         capacity,
118               VSID_ENDFIELD);
119   }
120
121   return ( rc );
122}
```


Notes

A table cannot store NULL pointers.

`VS_Table_AddEntry` determines the position in the table to store the new entry using a first available algorithm.

Table entries are zero-based (like arrays in C). The first entry in an empty table is stored in position 0, the second entry in a table is stored in position 1, and the nth entry in a table is stored in position n-1.

See Also

- VS_Table_Create(l),
- VS_Table_Destroy(l),
- VS_Table_GetFields(l),
- VS_Table_SetFields(l),
- VS_Table_CreateAddEntry(l),
- VS_Table_RemoveEntry(l)

VS_Table_Create

`VS_Table_Create` allocates a VolServ API table handle. A table handle is used to pass table information to and from VolServ.

A table is used to store a group of pointers of the same type within a single construct (much like an array or a linked list).

Synopsis

VST_TABLE_HANDLE VS_Table_Create
(VST_HANDLE_TYPE type,
int size)

Arguments

- `type` = The type of the pointers to be stored in the table. Valid TableCreate value types are enumerated in the *vs_types.h* file.
- `size` = The maximum number of entries to stored in the table.

Return Values

`VS_Table_Create` returns:

- A table handle, if one can be allocated.
- NULL, if a table handle cannot be allocated. An appropriate error code is set in `VSG_Error.`
- `VSE_ERR_OUTOFMEM` - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_createarchivemediaclass_execu
       te
4  *
5  * PURPOSE:
6  * This executes the
       VSCMD_CreateArchiveMediaClass
7  * API call.
8  *
9  * PARAMETERS:
10 * none
11
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
       vst_createarchivemediaclass_execu
       te(void)
15 #else
16    VST_BOOLEAN
       vst_createarchivemediaclass_execu
       te()
17 #endif
18 {
19    int                         i;
20    int                         count;
21    VST_BOOLEAN                 rc =
       VSE_FALSE;
22    VST_ARCHIVE_NAME            archive;
23    VST_MEDIA_CLASS_NAME
       mediaclass;
24    VST_CAPACITY                capacity;
25    VST_ARCHIVE_ACTION_OPTION   action;
26    VST_HIGH_MARK               highmark;
27    VST_LOW_MARK                lowmark;
28    VST_PRIORITY                migpri;
29    VST_ARCHIVE_NAME
       targetarchive;
```

```
30    VST_TABLE_HANDLE
         comphandletable;
31    VST_COMPONENT_HANDLE
         comphandle;
32    VST_COMP_TYPE  CompType =
         VSE_COMPTYPE_COLUMN;
33    ST_COMPONENT_ID            CompID;
34    VST_COMMAND_HANDLE         cmd;
35
36    bzero ( CompID, sizeof (
         VST_COMPONENT_ID ) );
37    /* get parameters from user */
38    printf("*** Create Archive Media
         Class parameters ***\n" );
39    printf("Enter Archive Name ==> " );
40    gets( archive );
41    printf("Enter Media Class Name ==> "
         );
42    gets( mediaclass );
43    printf("Enter Capacity Percent ==> "
         );
44    capacity = atoi(gets(input));
45    printf("Enter Archive action option
46       (0-none/1-mig/2-notify) ==> " );
47    action = atoi(gets(input));
48    printf("Enter High Mark Percentage
         ==> " );
49    highmark = atoi(gets(input));
50    printf("Enter Low Mark Percentage ==>
         " );
51    lowmark = atoi(gets(input));
52
53    if ( action == VSE_ARCHIVE_ACTION_MIG
         )
54    {
55       /* these parameters only need to
         be set */
56       /* if the archivemediaclass is
         being */
57       /* set up to support migration. */
58       printf("Enter Target Archive ==> "
         );
```

```
59          gets( targetarchive );
60          printf("Enter Migration Priority
            == > " );
61          migpri = atoi(gets(input));
62
            VSCMD_CreateArchiveMediaClass_Set
            Defaults (
63             VSID_TARGET_ARCHIVE_NAME,
            targetarchive,
64             VSID_MIGRATION_PRIORITY,
            migpri,
65             VSID_ENDFIELD );
66       }
67
68       printf("How many prefered placements
            (0 to skip): ");
69       count = atoi(gets(input));
70       if (count > 0)
71       {
72          comphandletable =
            VS_Table_Create(VSE_COMPONENT_HAN
            DLE, count);
73          if (comphandletable ==
            (VST_TABLE_HANDLE) NULL)
74          {
75             return(VSE_FALSE);
76          }
77          for (i = 0; i < count; i++)
78          {
79             printf("Enter row #%d:", i +
            1);
80             CompID[0] = (short)
            atoi(gets(input));
81             printf("Enter column #%d:", i +
            1);
82             CompID[1] = (short)
            atoi(gets(input));
83             CompID[2] = 0;
84             CompID[3] = 0;
85             comphandle =
            VS_Component_Create();
```

```
86
        VS_Component_SetFields(comphandle
        ,
87                      VSID_COMP_TYPE,
        CompType,
88                      VSID_COMP_ID,
        CompID,
89                      VSID_ENDFIELD);
90
        VS_Table_AddEntry(comphandletable
        ,comphandle);
91      }
92      /* This also only needs to be set
        if it is */
93      /* actually being used. */
94      /* It is not needed otherwise. */
95
        VSCMD_CreateArchiveMediaClass_Set
        Defaults(
96            VSID_COMPONENT_HANDLE_TABLE,
        comphandletable,
97            VSID_ENDFIELD);
98  }
99
100 /* create the command handle */
101 /* Note that the command handle is not
       destroyed*/
102 /* in this routine, but in
       vst_dispatch when */
103 /* final status is received. */
104 cmd = VS_Command_Create();
105 if (cmd != (VST_COMMAND_HANDLE )NULL)
106 {
107    /* Send the command to the VolServ
       software. */
108    /* Note that status is not
       processed here. */
109    /* Instead, it is processed in the
       vst_dispatch */
110    /* routine. Also, note that
       default values */
```

```
111      /* such as timeout, value retry
         limit, and */
112      /* priority are set as default
         parameters. */
113      rc =
         VSCMD_CreateArchiveMediaClass(cmd
         ,
114               VSID_ARCHIVE_NAME,
         archive,
115               VSID_MEDIA_CLASS_NAME,
         mediaclass,
116               VSID_HIGH_MARK,
         highmark,
117               VSID_LOW_MARK,
         lowmark,
118               VSID_CAPACITY,
         capacity,
119               VSID_ENDFIELD);
120   }
121
122   return ( rc );
123}
```

Notes

Client adds entries with the `VS_Table_AddEntry`, `VS_Table_SetFields`, and/or `VS_Table_CreateAddEntry` functions.

When a client adds an entry to a table using the `VS_Table_AddEntry` or `VS_Table_SetFields` function, the client is responsible for allocating and maintaining the space to contain the information maintained in the table.

When a client adds an entry to a table using the `VS_Table_CreateAddEntry` function, the API allocates space for a table entry, copies the client's data to the allocated space, and adds the entry to the specified table. After calling `VS_Table_CreateAddEntry`, a client is no longer required to maintain the space containing the information added to the table.

Client deletes entries from a table with the `VS_Table_RemoveEntry` function. If the entry to be deleted from the table was added with either the `VS_Table_AddEntry` or the `VS_Table_SetFields` function, it is the client's responsibility to free the space for the entry after it has been removed from the table. If the entry to be deleted from the table was added with the `VS_Table_CreateAddEntry` function, the API frees the space for the entry after it has been removed from the table.

See Also

- VS_Table_Destroy(l),
- VS_Table_GetFields(l),
- VS_Table_SetFields(l),
- VS_Table_AddEntry(l),
- VS_Table_CreateAddEntry(l),
- VS_Table_RemoveEntry(l)

VS_Table_CreateAdd-Entry

`VS_Table_CreateAddEntry` adds an entry to the specified table at the first available location.

When a client adds an entry to a table using `VS_Table_CreateAddEntry`, the VolServ API allocates space for a table entry, copies the client's data to the allocated space, and adds an entry to the specified table for the copied data. After calling `VS_Table_CreateAddEntry`, a client is not required to maintain the information that has been stored in the table.

Synopsis

```
VST_BOOLEAN VS_Table_CreateAddEntry
(VST_TABLE_HANDLE handle,
void * entry,
int size)
```

Arguments

- `handle =` Handle of the table where the specified entry is added.
- `entry` = Pointer to the entry to be copied and added to the specified table.
- `size` = The size, in bytes, of the entry to be copied and added to the specified table.

Return Values

`VS_Table_CreateAddEntry` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADENTRY` - Pointer to the entry to be stored was null.

- VSE_ERR_BADHANDLE - Specified handle was not a table handle.
- VSE_ERR_FULL - The table is full and the entry could not be added
- VSE_ERR_NULLHANDLE -Specified handle was a null pointer.
- VSE_ERR_OUTOFMEM - The allocation request for space to copy the client's data failed.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_table_handle
4  *
5  * PURPOSE:
6  * This function tests the table handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_table_handle(void)
14 #else
15    VST_BOOLEAN vst_table_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_TRUE;
19    VST_TABLE_HANDLE        tableh;
20    int                     numdrives;
21    int                     i;
22    VST_DRIVE_ID            testdriveid
         = 9999;
23    int                     numentries;
24    VST_DRIVE_ID            * ip;
25
```

```
26    printf("*** Table Handle test
         ***\n");
27    printf("How many drives? ");
28    numdrives = atoi(gets(input));
29
30    /* create the table handle */
31    tableh =
         VS_Table_Create(VSE_INTEGER_PTR,
         numdrives);
32
33    /* generate drive ids */
34    for (i = 0; i < numdrives; i++)
35    {
36       VS_Table_CreateAddEntry(tableh,
         &i, sizeof(VST_DRIVE_ID));
37    }
38    /* get the number of entries in the
         table */
39    VS_Table_GetFields(tableh,
40                  VSID_NUMBER_ENTRIES,
         &numentries,
41                  VSID_ENDFIELD);
42    /* loop through the table and print
         the entries */
43    for (i = 0; i < numentries; i++)
44    {
45       VS_Table_GetFields(tableh,
46                     VSID_TABLE_ENTRY,
         i, &ip,
47                     VSID_ENDFIELD);
48       /* remove the entry from the table
         */
49       VS_Table_RemoveEntry(tableh, ip);
50       /* set this entry to a NULL
         pointer */
51       VS_Table_SetFields(tableh,
52
         VSID_TABLE_ENTRY, i, NULL,
53                        VSID_ENDFIELD);
54       /* print the value retrieved from
         the table */
```

```
55       printf("Drive ID #%d = %d\n", i,
         *ip);
56    }
57    /* destroy the table */
58    VS_Table_Destroy(tableh);
59    return(rc);
60 }
```

Notes

A table cannot store NULL pointers.

`VS_Table_CreateAddEntry` determines the location in the table to store the entry using a first available algorithm.

Table entries are zero-based (like arrays in C). The first entry in an empty table is stored in position 0, the second entry in a table is stored in position 1, and the n^{th} entry in a table is stored in position n-1.

See Also

- VS_Table_AddEntry(l),
- VS_Table_Create(l),
- VS_Table_Destroy(l),
- VS_Table_GetFields(l),
- VS_Table_RemoveEntry(l),
- VS_Table_SetFields(l)

VS_Table_Destroy

VS_Table_Destroy deallocates a VolServ API table handle that was allocated with the VS_Table_Create function. A table handle is used to pass table information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Table_Destroy
(VST_TABLE_HANDLE handle)

Arguments

- handle = The table handle to be destroyed.

Return Values

VS_Table_Destroy returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
- VSE_ERR_BADHANDLE - Specified handle was not a table handle.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_table_handle
4  *
5  * PURPOSE:
6  * This function tests the table handle.
7  *
8  * PARAMETERS:
9  * none
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_table_handle(void)
14 #else
15    VST_BOOLEAN vst_table_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_TRUE;
19    VST_TABLE_HANDLE        tableh;
20    int                     numdrives;
21    int                     i;
22    VST_DRIVE_ID            testdriveid
         = 9999;
23    int                     numentries;
24    VST_DRIVE_ID            * ip;
25
26    printf("*** Table Handle test
         ***\n");
27    printf("How many drives? ");
28    numdrives = atoi(gets(input));
29
30    /* create the table handle */
31    tableh =
         VS_Table_Create(VSE_INTEGER_PTR,
         numdrives);
32
33    /* generate drive ids */
34    for (i = 0; i < numdrives; i++)
35    {
36       VS_Table_CreateAddEntry(tableh,
         &i, sizeof(VST_DRIVE_ID));
37    }
38    /* get the number of entries in the
         table */
39    VS_Table_GetFields(tableh,
40                  VSID_NUMBER_ENTRIES,
         &numentries,
41                  VSID_ENDFIELD);
42    /* loop through the table and print
         the entries */
```

```
43    for (i = 0; i < numentries; i++)
44    {
45       VS_Table_GetFields(tableh,
46
         VSID_TABLE_ENTRY, i, &ip,
47                         VSID_ENDFIELD);
48       /* remove the entry from the table
         */
49       VS_Table_RemoveEntry(tableh, ip);
50       /* set this entry to a NULL
         pointer */
51       VS_Table_SetFields(tableh,
52
         VSID_TABLE_ENTRY, i, NULL,
53                         VSID_ENDFIELD);
54       /* print the value retrieved from
         the table */
55       printf("Drive ID #%d = %d\n", i,
         *ip);
56    }
57    /* destroy the table */
58    VS_Table_Destroy(tableh);
59    return(rc);
60 }
```

Notes

After `VS_Table_Destroy` has been called for a table handle, that handle is no longer valid and should not be used.

If a client adds entries to a table using `VS_Table_AddEntry` or `VS_Table_SetFields` functions, it is the client's responsibility to maintain the data after it is stored in the table. It is also the client's responsibility to deallocate the space containing the data after the entry has been deleted from the table and is no longer needed.

If a client adds entries to a table using `VS_Table_CreateAddEntry`, the VolServ API maintains the data that is stored in the table and frees the space allocated for all table entries when the `VS_Table_Destroy` function is called.

See Also

- VS_Table_AddEntry(l),
- VS_Table_Create(l),
- VS_Table_CreateAddEntry(l),
- VS_Table_GetFields(l),
- VS_Table_RemoveEntry(l),
- VS_Table_SetFields(l)

VS_Table_Get Fields

`VS_Table_GetFields` retrieves information associated with a table handle. A table handle is used to pass table information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Table_GetFields
(VST_TABLE_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Table handle where information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_NUMBER_ENTRIES (int *)	Pointer to the number of entries in the table.
VSID_TABLE_ENTRY (int)	Index of a specific entry in the table.
(void **)	Pointer to a specific entry in the table.

Return Values

`VS_Table_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a request handle.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.
- `VSE_ERR_OUTOFRANGE` - Specified entry does not exist in the table's range of values.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_print_drivepool
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored in
7  * a drive pool handle.
8  *
9  * PARAMETERS:
10 * h : the drive pool handle to print
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
```

```
14    void
         vst_print_drivepool(VST_DRIVEPOOL
         _HANDLE h)
15 #else
16    void vst_print_drivepool(h)
17    VST_DRIVEPOOL_HANDLE h;
18 #endif
19 {
20    VST_DRIVE_POOL_NAME  DrivePoolName;
21    VST_TABLE_HANDLE
         DriveHandleTable;
22    VST_DRIVE_HANDLE     DriveHandle;
23    int                  i;
24    int                  n;
25
26    VS_DrivePool_GetFields(h,
27       VSID_DRIVEPOOL_NAME,
         DrivePoolName,
28       VSID_DRIVE_HANDLE_TABLE,
         &DriveHandleTable,
29       VSID_ENDFIELD);
30    printf("DrivePoolName =
         %s\n",DrivePoolName);
31    /* Get # of entries */
32    if ( DriveHandleTable !=
         (VST_TABLE_HANDLE) NULL )
33    {
34
         VS_Table_GetFields(DriveHandleTab
         le,
35
         VSID_NUMBER_ENTRIES, &n,
36                       VSID_ENDFIELD);
37       for ( i = 0; i < n; i++)
38       {
39
         VS_Table_GetFields(DriveHandleTab
         le,
40                VSID_TABLE_ENTRY, i,
         &DriveHandle,
41                VSID_ENDFIELD);
42          vst_print_drive(DriveHandle);
```

```
43        }
44     }
45 }
```

Notes

A table is a VolServ API structure that holds a group of like pointers. Table handles are used to return lists of information to the client software. `VS_Table_GetFields` allows the user to access the entries in the table.

The `VSID_TABLE_ENTRY` parameter requires that two arguments be passed instead of one. The first argument is the index of the entry in the appropriate table. The second argument is a pointer to a location where the value is stored.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Archive_GetFields(l),
- VS_Criteria_GetFields(l),
- VS_CriteriaGroup_GetFields(l),
- VS_Drive_GetFields(l),
- VS_DrivePool_GetFields(l),
- VS_Notify_GetFields(l),
- VS_Status_GetFields(l),
- VS_Table_AddEntry(l),
- VS_Table_Create(l),
- VS_Table_CreateAddEntry(l),
- VS_Table_Destroy(l),
- VS_Table_RemoveEntry(l),
- VS_Table_SetFields(l)

VS_Table_ RemoveEntry

`VS_Table_RemoveEntry` removes an entry from the specified VolServ API table. A table handle is used to pass table information to and from VolServ.

Synopsis

VST_BOOLEAN VS_Table_RemoveEntry
(VST_TABLE_HANDLE handle,
void * entry)

Arguments

- `handle` = Handle of the table where the entry is removed.
- `entry` = Pointer to the entry to be removed from the table.

Return Values

`VS_Table_RemoveEntry` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error.`
- `VSE_ERR_BADENTRY` - Pointer to the entry to be removed was null
- `VSE_ERR_BADHANDLE` - Specified handle was not a table handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_table_handle
4  *
5  * PURPOSE:
6  * This function tests the table handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_table_handle(void)
14 #else
15    VST_BOOLEAN vst_table_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_TRUE;
19    VST_TABLE_HANDLE        tableh;
20    int                     numdrives;
21    int                     i;
22    VST_DRIVE_ID            testdriveid
         = 9999;
23    int                     numentries;
24    VST_DRIVE_ID            * ip;
25
26    printf("*** Table Handle test
         ***\n");
27    printf("How many drives? ");
28    numdrives = atoi(gets(input));
29
30    /* create the table handle */
31    tableh =
         VS_Table_Create(VSE_INTEGER_PTR,
         numdrives);
32
33    /* generate drive ids */
34    for (i = 0; i < numdrives; i++)
35    {
```

```
36       VS_Table_CreateAddEntry(tableh,
         &i, sizeof(VST_DRIVE_ID));
37    }
38    /* get the number of entries in the
         table */
39    VS_Table_GetFields(tableh,
40                  VSID_NUMBER_ENTRIES,
         &numentries,
41                  VSID_ENDFIELD);
42    /* loop through the table and print
         the entries */
43    for (i = 0; i < numentries; i++)
44    {
45       VS_Table_GetFields(tableh,
46
         VSID_TABLE_ENTRY, i, &ip,
47                         VSID_ENDFIELD);
48       /* remove the entry from the table
         */
49       VS_Table_RemoveEntry(tableh, ip);
50       /* set this entry to a NULL
         pointer */
51       VS_Table_SetFields(tableh,
52
         VSID_TABLE_ENTRY, i, NULL,
53                         VSID_ENDFIELD);
54       /* print the value retrieved from
         the table */
55       printf("Drive ID #%d = %d\n", i,
         *ip);
56    }
57    /* destroy the table */
58    VS_Table_Destroy(tableh);
59    return(rc);
60 }
```

Notes

`VS_Table_RemoveEntry` verifies that `entry` (a pointer value) matches an entry in the specified table. (Note that all table entries are pointers to data.) `VS_Table_RemoveEntry` does not verify that the data pointed to by `entry` matches the data pointed to by the corresponding table entry. As long as the pointer values match, `VS_Table_RemoveEntry` removes the entry from the table.

If a client adds entries to a table using `VS_Table_AddEntry` or `VS_Table_SetFields` functions, it is the client's responsibility to maintain the data after it is stored in the table. It is also the client's responsibility to deallocate the space containing the data after the entry has been deleted from the table and is no longer needed.

If a client adds entries to a table using `VS_Table_CreateAddEntry`, the VolServ API maintains the data that is stored in the table and frees the space allocated for all table entries when the `VS_Table_Destroy` function is called.

See Also

- VS_Table_AddEntry(l),
- VS_Table_Create(l),
- VS_Table_CreateAddEntry(l),
- VS_Table_Destroy(l),
- VS_Table_GetFields(l),
- VS_Table_SetFields(l)

VS_Table_SetFields

`VS_Table_SetFields` adds an entry to the specified table at the specified location.

When a client adds an entry to a table using `VS_Table_SetFields`, the client is responsible for allocating and maintaining the space to contain the information maintained in the table.

Synopsis

VST_BOOLEAN VS_Table_GetFields
(VST_TABLE_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = Handle of the table where an entry is added.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
SID_NUMBER_ENTRIES (int)	Number of entries in the table.
VSID_TABLE_ENTRY (int)	Position where the specified entry is added to the table.
(void *)	Pointer to the entry to be stored in the table.

Return Values

`VS_Table_GetFields` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADHANDLE` - Specified handle was not a request handle.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.
- `VSE_ERR_OUTOFRANGE` - Specified entry does not exist in the table's range of values.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_table_handle
4  *
5  * PURPOSE:
6  * This function tests the table handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_table_handle(void)
14 #else
```

```
15    VST_BOOLEAN vst_table_handle()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_TRUE;
19    VST_TABLE_HANDLE        tableh;
20    int                     numdrives;
21    int                     i;
22    VST_DRIVE_ID            testdriveid
         = 9999;
23    int                     numentries;
24    VST_DRIVE_ID            * ip;
25
26    printf("*** Table Handle test
         ***\n");
27    printf("How many drives? ");
28    numdrives = atoi(gets(input));
29
30    /* create the table handle */
31    tableh =
         VS_Table_Create(VSE_INTEGER_PTR,
         numdrives);
32
33    /* generate drive ids */
34    for (i = 0; i < numdrives; i++)
35    {
36       VS_Table_CreateAddEntry(tableh,
         &i, sizeof(VST_DRIVE_ID));
37    }
38    /* get the number of entries in the
         table */
39    VS_Table_GetFields(tableh,
40                  VSID_NUMBER_ENTRIES,
         &numentries,
41                  VSID_ENDFIELD);
42    /* loop through the table and print
         the entries */
43    for (i = 0; i < numentries; i++)
44    {
45       VS_Table_GetFields(tableh,
46
         VSID_TABLE_ENTRY, i, &ip,
```

```
47                          VSID_ENDFIELD);
48        /* remove the entry from the table
          */
49        VS_Table_RemoveEntry(tableh, ip);
50        /* set this entry to a NULL
          pointer */
51        VS_Table_SetFields(tableh,
52
          VSID_TABLE_ENTRY, i, NULL,
53                          VSID_ENDFIELD);
54        /* print the value retrieved from
          the table */
55        printf("Drive ID #%d = %d\n", i,
          *ip);
56     }
57     /* destroy the table */
58     VS_Table_Destroy(tableh);
59     return(rc);
60  }
```


Notes

A table cannot store NULL pointers.

Table entries are zero-based (like arrays in C). The first entry in an empty table is stored in position 0, the second entry in a table is stored in position 1, and the nth entry in a table is stored in position n-1.

The `VSID_TABLE_ENTRY` parameter requires that two arguments be passed instead of one. The first argument is the position where the specified entry is added to the table. The second argument is the entry to be added to the table.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Archive_GetFields(l),
- VS_DrivePool_GetFields(l),
- VS_Error_GetFields(l),
- VS_MediaClass_Getfields(l),
- VS_Table_AddEntry(l),
- VS_Table_Create(l),
- VS_Table_CreateAddEntry(l),
- VS_Table_Destroy(l),
- VS_Table_GetFields(l),
- VS_Table_RemoveEntry(l)

VS_Terminate

`VS_Terminate` terminates the VolServ API software. `VS_Terminate` releases all memory allocated by the API and deregisters and frees all RPC transports.

Synopsis

VST_BOOLEAN VS_Terminate (void)

Arguments

None

Return Values

`VS_Terminate` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_NOTINITIALIZED` - The VolServ API is not initialized.
- `VSE_ERR_SYSTEMCALL` - A system call failed (usually from RPC). This generic error code covers an error that stems from a system call. The API sets this when encountering a failure during RPC setup.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_terminate
4  *
5  * PURPOSE:
6  * This function shuts down the VolServ
       API.
7  *
8  * PARAMETERS:
9  * none
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_terminate ( void )
14 #else
15    VST_BOOLEAN vst_terminate ()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19
20    rc = VS_Terminate();
21    return(rc);
22 }
```

Notes

All command functions fail after `VS_Terminate` has been invoked.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Initialize(l)

VS_TypeCapacity_Create

VS_TypeCapacity_Create allocates a VolServ API archive type capacity handle. A type capacity handle is used to pass archive type capacity information from VolServ in response to an Archive Query command request with the type capacity option specified.

Synopsis

VST_TYPECAPACITY_HANDLE VS_TypeCapacity_Create (void)

Arguments

None

Return Values

VS_TypeCapacity_Create returns:

- A type capacity handle, if one can be allocated.
- NULL, if a type capacity handle could not be allocated. An appropriate error code is set in VSG_Error.
- VSE_ERR_OUTOFMEM - Memory allocation error.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_typecapacity_handle
4  *
5  * PURPOSE:
6  * This function tests a typecapacity
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       **********/
12 #ifdef ANSI_C
```

```
13    VST_BOOLEAN
         vst_typecapacity_handle(void)
14 #else
15    VST_BOOLEAN
         vst_typecapacity_handle()
16 #endif
17 {
18    VST_BOOLEAN                rc =
         VSE_FALSE;
19    VST_TYPECAPACITY_HANDLE    h;
20    VST_MEDIA_TYPE_NAME
         MediaType;
21    VST_CAPACITY               Capacity;
22    VST_HIGH_MARK
         HighMarkPercent;
23    VST_LOW_MARK
         LowMarkPercent;
24    VST_FILL_LEVEL
         FillLevel;
25    int AssignedBins;
26    VST_ARCHIVE_ACTION_OPTION
         ActionOpt;
27    VST_BOOLEAN
         AutoCheckinFlag;
28    VST_BOOLEAN
         AutoImportFlag;
29    VST_BATCH_NAME
         ImportBatch;
30    VST_MANUFACTURER_NAME
         ImportManufacturer;
31    VST_MEDIA_CLASS_NAME
         ImportClass;
32
33    /* create the handle */
34    h = vS_TypeCapacity_Create();
35    if (h != (VST_TYPECAPACITY_HANDLE)
         NULL)
36    {
37       /* get values from user */
38       printf("Enter media type name ==>
         ");
39       gets(MediaType);
```

```
40        printf("Enter import class ==> ");
41        gets(ImportClass);
42        printf("Enter import batch ==> ");
43        gets(ImportBatch);
44        printf("Enter import manufacturer
          ==> ");
45        gets(ImportManufacturer);
46        printf("Enter capacity ==> ");
47        Capacity = atoi(gets(input));
48        printf("Enter high mark percent
          ==> ");
49        HighMarkPercent =
          atoi(gets(input));
50        printf("Enter low mark percent ==>
          ");
51        LowMarkPercent =
          atoi(gets(input));
52        printf("Enter fill level ==> ");
53        FillLevel = atoi(gets(input));
54        printf("Enter number of assigned
          bins ==> ");
55        AssignedBins = atoi(gets(input));
56        printf("Enter archive action
          option ==> ");
57        ActionOpt = atoi(gets(input));
58        printf("Enter auto import flag ==>
          ");
59        AutoImportFlag =
          atoi(gets(input));
60        printf("Enter auto checkin flag
          ==> ");
61        AutoCheckinFlag =
          atoi(gets(input));
62        /* set the fields */
63        rc = VS_TypeCapacity_SetFields(h,
64           VSID_MEDIA_TYPE_NAME,
          MediaType,
65           VSID_MEDIA_CLASS_NAME,
          ImportClass,
66           VSID_BATCH_NAME,
          ImportBatch,
```

```
67          VSID_MANUFACTURER,
          ImportManufacturer,
68          VSID_CAPACITY,
          Capacity,
69          VSID_HIGH_MARK,
          HighMarkPercent,
70          VSID_LOW_MARK,
          LowMarkPercent,
71          VSID_FILL_LEVEL,
          FillLevel,
72          VSID_ASSIGNED_BINS,
          AssignedBins,
73          VSID_ARCHIVE_ACTION,
          ActionOpt,
74          VSID_AUTOIMPORT_FLAG,
          AutoImportFlag,
75          VSID_AUTOCHECKIN_FLAG,
          AutoCheckinFlag,
76          VSID_ENDFIELD);
77       if (rc)
78       {
79          vst_print_typecapacity(h);
80       }
81       VS_TypeCapacity_Destroy(h);
82    }
83    return(rc);
84 }
```

Notes

None

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_TypeCapacity_Destroy(l),
- VS_TypeCapacity_GetFields(l),
- VS_TypeCapacity_SetFields(l)

VS_TypeCapacity_Destroy

VS_TypeCapacity_Destroy deallocates a type capacity handle that was allocated by VS_TypeCapacity_Create. A type capacity handle is used to pass archive type capacity information from VolServ in response to an Archive Query command request with the type capacity option specified.

Synopsis

VST_BOOLEAN VS_TypeCapacity_Destroy
(VST_TYPECAPACITY_HANDLE handle)

Arguments

- handle = The type capacity handle to be destroyed.

Return Values

VS_TypeCapacity_Destroy returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADHANDLE - Specified handle was not a type capacity handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_typecapacity_handle
4  *
5  * PURPOSE:
6  * This function tests a typecapacity
       handle.
7  *
8  * PARAMETERS:
9  * none
```

```
10 *
11 ************************************
        *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_typecapacity_handle(void)
14 #else
15    VST_BOOLEAN
         vst_typecapacity_handle()
16 #endif
17 {
18    VST_BOOLEAN                rc =
         VSE_FALSE;
19    VST_TYPECAPACITY_HANDLE    h;
20    VST_MEDIA_TYPE_NAME
         MediaType;
21    VST_CAPACITY               Capacity;
22    VST_HIGH_MARK
         HighMarkPercent;
23    VST_LOW_MARK
         LowMarkPercent;
24    VST_FILL_LEVEL
         FillLevel;
25    int AssignedBins;
26    VST_ARCHIVE_ACTION_OPTION
         ActionOpt;
27    VST_BOOLEAN
         AutoCheckinFlag;
28    VST_BOOLEAN
         AutoImportFlag;
29    VST_BATCH_NAME
         ImportBatch;
30    VST_MANUFACTURER_NAME
         ImportManufacturer;
31    VST_MEDIA_CLASS_NAME
         ImportClass;
32
33    /* create the handle */
34    h = vS_TypeCapacity_Create();
35    if (h != (VST_TYPECAPACITY_HANDLE)
         NULL)
36    {
```

```
37       /* get values from user */
38       printf("Enter media type name ==>
         ");
39       gets(MediaType);
40       printf("Enter import class ==> ");
41       gets(ImportClass);
42       printf("Enter import batch ==> ");
43       gets(ImportBatch);
44       printf("Enter import manufacturer
         ==> ");
45       gets(ImportManufacturer);
46       printf("Enter capacity ==> ");
47       Capacity = atoi(gets(input));
48       printf("Enter high mark percent
         ==> ");
49       HighMarkPercent =
         atoi(gets(input));
50       printf("Enter low mark percent ==>
         ");
51       LowMarkPercent =
         atoi(gets(input));
52       printf("Enter fill level ==> ");
53       FillLevel = atoi(gets(input));
54       printf("Enter number of assigned
         bins ==> ");
55       AssignedBins = atoi(gets(input));
56       printf("Enter archive action
         option ==> ");
57       ActionOpt = atoi(gets(input));
58       printf("Enter auto import flag ==>
         ");
59       AutoImportFlag =
         atoi(gets(input));
60       printf("Enter auto checkin flag
         ==> ");
61       AutoCheckinFlag =
         atoi(gets(input));
62       /* set the fields */
63       rc = VS_TypeCapacity_SetFields(h,
64          VSID_MEDIA_TYPE_NAME,
         MediaType,
```

```
65          VSID_MEDIA_CLASS_NAME,
          ImportClass,
66          VSID_BATCH_NAME,
          ImportBatch,
67          VSID_MANUFACTURER,
          ImportManufacturer,
68          VSID_CAPACITY,
          Capacity,
69          VSID_HIGH_MARK,
          HighMarkPercent,
70          VSID_LOW_MARK,
          LowMarkPercent,
71          VSID_FILL_LEVEL,
          FillLevel,
72          VSID_ASSIGNED_BINS,
          AssignedBins,
73          VSID_ARCHIVE_ACTION,
          ActionOpt,
74          VSID_AUTOIMPORT_FLAG,
          AutoImportFlag,
75          VSID_AUTOCHECKIN_FLAG,
          AutoCheckinFlag,
76          VSID_ENDFIELD);
77       if (rc)
78       {
79          vst_print_typecapacity(h);
80       }
81       VS_TypeCapacity_Destroy(h);
82    }
83    return(rc);
84 }
```

Notes

After `VS_TypeCapacity_Destroy` has been called for a type capacity handle, that handle is no longer valid and should not be used.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_TypeCapacity_Create(l),
- VS_TypeCapacity_GetFields(l),
- VS_TypeCapacity_SetFields(l)

VS_ TypeCapacity _GetFields

VS_TypeCapacity_GetFields retrieves information from a type capacity handle. A type capacity handle is used to pass archive type capacity information from VolServ in response to an Archive Query command request with the type capacity option specified.

Synopsis

VST_BOOLEAN VS_TypeCapacity_GetFields
(VST_TYPECAPACITY_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- handle = The type capacity handle where information is retrieved.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by a pointer to a location where the value of the parameter may be stored. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_ACTION (VST_ARCHIVE_ACTION_OPTION*)	Pointer to the action taken by VolServ when the number of media of this media type classification reaches the high mark threshold (migrate, notify, or none) in the archive. Valid VSID_ARCHIVE_ACTION values are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_ASSIGNED_BINS (VST_COUNT *)	Pointer to the number of bins to be assigned to this media type classification within the archive.
VSID_AUTOIMPORT_FLAG (VST_BOOLEAN *)	Pointer to a flag indicating whether automatic import is allowable for the archive.
VSID_AUTOCHECKIN_FLAG (VST_BOOLEAN *)	Pointer to a flag indicating whether automatic checkin is active for the archive.
VSID_BATCH_NAME (VST_BATCH_NAME)	Pointer to the batch name to be assigned to automatically imported/checked in media. Valid batch names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CAPACITY (VST_CAPACITY *)	Pointer to the maximum number of media of this media type classification that can be in this archive.
VSID_FILL_LEVEL (VST_FILL_LEVEL *)	Pointer to the current number of media of this media type classification in this archive.
VSID_HIGH_MARK (VST_HIGH_MARK *)	Pointer to the percentage of `VSID_CAPACITY` above which the specified migration policy option is performed or initiated.
VSID_LOW_MARK (VST_LOW_MARK *)	Pointer to the percentage of `VSID_CAPACITY` below which the specified migration policy option is performed or terminated.
VSID_MANUFACTURER (VST_MANUFACTURER_NAME)	Pointer to the manufacturer to be assigned to automatically imported/checked in media. Valid manufacturer names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS)	Pointer to the MediaClass classification where automatically imported/checked in media are assigned.

Parameter Type	Description
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	Pointer to the name of this media type classification. Valid media type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Return Values

`VS_TypeCapacity_GetField` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_print_typecapacity
4  *
5  * PURPOSE:
6  * This function prints out the
       information stored
7  * in a type capacity handle.
8  *
9  * PARAMETERS:
10 * h : the type capacity handle to print
11 *
12 **************************************
       *********/
```

```
13 #ifdef ANSI_C
14    void
         vst_print_typecapacity(VST_TYPECA
         PACITY_HANDLEh)
15 #else
16    void vst_print_typecapacity(h)
17    VST_TYPECAPACITY_HANDLE h;
18 #endif
19 {
20    VST_MEDIA_TYPE_NAME
         MediaType;
21    VST_CAPACITY                Capacity;
22    VST_HIGH_MARK
         HighMarkPercent;
23    VST_LOW_MARK
         LowMarkPercent;
24    VST_FILL_LEVEL
         FillLevel;
25    int
         AssignedBins;
26    VST_ARCHIVE_ACTION_OPTION
         ActionOpt;
27    VST_BOOLEAN
         AutoCheckinFlag;
28    VST_BOOLEAN
         AutoImportFlag;
29    VST_BATCH_NAME
         ImportBatch;
30    VST_MANUFACTURER_NAME
         ImportManufacturer;
31    VST_MEDIA_CLASS_NAME
         ImportClass;
32
33    VS_TypeCapacity_GetFields(h,
34          VSID_MEDIA_TYPE_NAME,
         MediaType,
35          VSID_MEDIA_CLASS_NAME,
         ImportClass,
36          VSID_BATCH_NAME,
         ImportBatch,
37          VSID_MANUFACTURER,
         ImportManufacturer,
```

```
38          VSID_CAPACITY,
          &Capacity,
39          VSID_HIGH_MARK,
          &HighMarkPercent,
40          VSID_LOW_MARK,
          &LowMarkPercent,
41          VSID_FILL_LEVEL,
          &FillLevel,
42          VSID_ASSIGNED_BINS,
          &AssignedBins,
43          VSID_ARCHIVE_ACTION,
          &ActionOpt,
44          VSID_AUTOIMPORT_FLAG,
          &AutoImportFlag,
45          VSID_AUTOCHECKIN_FLAG,
          &AutoCheckinFlag,
46          VSID_ENDFIELD);
47    printf("******* Type Capacity Handle
          *******\n");
48    printf("Media type = %s\n",
          MediaType);
49    printf("Capacity = %d\n", Capacity);
50    printf("High Mark Percent = %d\n",
          HighMarkPercent);
51    printf("Low Mark Percent = %d\n",
          LowMarkPercent);
52    printf("Fill Level = %d\n",
          FillLevel);
53    printf("Assigned Bins = %d\n",
          AssignedBins);
54    printf("Archive Action Option =
          %d\n", ActionOpt);
55    printf("Auto Checkin = %d\n",
          AutoCheckinFlag);
56    printf("Auto Import = %d\n",
          AutoImportFlag);
57    printf("Import Class = %s\n",
          ImportClass);
58    printf("Import Batch = %s\n",
          ImportBatch);
59    printf("Import Manufacturer = %s\n",
          ImportManufacturer);
```

```
60 }
```

Notes

The migration policy options for `VSID_HIGH_MARK` are no action, operator notification, and automatic migration.

When the number of media in a media type classification reaches the high mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy option is set to notify.
- Initiates automatic migration of media if the migration policy is set to migrate.

When the number of media in a media type classification drops to the low mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy is set to notify.
- Terminates automatic migration of media if the migration policy is set to migrate.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Archive_GetFields(l),
- VS_Error_GetFields(l),
- VS_TypeCapacity_Create(l),
- VS_TypeCapacity_Destroy(l),
- VS_TypeCapacity_SetFields(l),
- VSCMD_ArchiveQuery(l)

VS_ TypeCapacity _SetFields

`VS_TypeCapacity_SetFields` sets the value of one or more fields in a type capacity handle. A type capacity handle is used to pass archive type capacity information to and from VolServ.

Synopsis

VST_BOOLEAN VS_TypeCapacity_SetFields
(VST_TYPECAPACITY_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The type capacity handle where information is stored or updated.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to store. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_ACTION (VST_ARCHIVE_ACTION_OPTION)	Action taken by VolServ when the number of media of this media type classification reaches the high mark threshold (migrate, notify, or none). Valid `VSID_ARCHIVE_ACTION` values are enumerated in the `vs_types.h` file.
VSID_ASSIGNED_BINS (VST_COUNT)	Number of bins to be assigned to the media type classification within the archive.

Parameter Type	Description
VSID_AUTOCHECKIN_FLAG (VST_BOOLEAN)	Flag indicating whether automatic checkin is active for the archive.
VSID_AUTOIMPORT_FLAG (VST_BOOLEAN)	Flag indicating whether automatic import is active for the archive.
VSID_BATCH_NAME (VST_BATCH_NAME)	Batch name to be assigned to automatically imported/checked in media. Valid batch names may contain 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CAPACITY (VST_CAPACITY)	Maximum number of media of this media type classification that can be in this archive.
VSID_FILL_LEVEL (VST_FILL_LEVEL)	Current number of media of this media type classification in this archive.
VSID_HIGH_MARK (VST_HIGH_MARK)	Percentage of VSID_CAPACITY above which the specified migration policy option is performed or initiated.
VSID_LOW_MARK (VST_LOW_MARK)	Percentage of VSID_CAPACITY below which the specified migration policy option is performed or terminated.
VSID_MANUFACTURER (VST_MANUFACTURER_NAME)	Manufacturer to be assigned to automatically imported/checked in media. Valid manufacturer names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS)	MediaClass group where automatically imported/checked in media are assigned.
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	Name of this media type classification. Valid media type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Return Values

VS_TypeCapacity_SetFields returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADHANDLE - Specified handle was not a type capacity handle.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_typecapacity_handle
4  *
5  * PURPOSE:
6  * This function tests a typecapacity
       handle.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_typecapacity_handle(void)
14 #else
```

```
15    VST_BOOLEAN
         vst_typecapacity_handle()
16 #endif
17 {
18    VST_BOOLEAN                rc =
         VSE_FALSE;
19    VST_TYPECAPACITY_HANDLE    h;
20    VST_MEDIA_TYPE_NAME
         MediaType;
21    VST_CAPACITY               Capacity;
22    VST_HIGH_MARK
         HighMarkPercent;
23    VST_LOW_MARK
         LowMarkPercent;
24    VST_FILL_LEVEL
         FillLevel;
25    int AssignedBins;
26    VST_ARCHIVE_ACTION_OPTION
         ActionOpt;
27    VST_BOOLEAN
         AutoCheckinFlag;
28    VST_BOOLEAN
         AutoImportFlag;
29    VST_BATCH_NAME
         ImportBatch;
30    VST_MANUFACTURER_NAME
         ImportManufacturer;
31    VST_MEDIA_CLASS_NAME
         ImportClass;
32
33    /* create the handle */
34    h = vS_TypeCapacity_Create();
35    if (h != (VST_TYPECAPACITY_HANDLE)
         NULL)
36    {
37       /* get values from user */
38       printf("Enter media type name ==>
         ");
39       gets(MediaType);
40       printf("Enter import class ==> ");
41       gets(ImportClass);
42       printf("Enter import batch ==> ");
```

```
43        gets(ImportBatch);
44        printf("Enter import manufacturer
          ==> ");
45        gets(ImportManufacturer);
46        printf("Enter capacity ==> ");
47        Capacity = atoi(gets(input));
48        printf("Enter high mark percent
          ==> ");
49        HighMarkPercent =
          atoi(gets(input));
50        printf("Enter low mark percent ==>
          ");
51        LowMarkPercent =
          atoi(gets(input));
52        printf("Enter fill level ==> ");
53        FillLevel = atoi(gets(input));
54        printf("Enter number of assigned
          bins ==> ");
55        AssignedBins = atoi(gets(input));
56        printf("Enter archive action
          option ==> ");
57        ActionOpt = atoi(gets(input));
58        printf("Enter auto import flag ==>
          ");
59        AutoImportFlag =
          atoi(gets(input));
60        printf("Enter auto checkin flag
          ==> ");
61        AutoCheckinFlag =
          atoi(gets(input));
62        /* set the fields */
63        rc = VS_TypeCapacity_SetFields(h,
64           VSID_MEDIA_TYPE_NAME,
          MediaType,
65           VSID_MEDIA_CLASS_NAME,
          ImportClass,
66           VSID_BATCH_NAME,
          ImportBatch,
67           VSID_MANUFACTURER,
          ImportManufacturer,
68           VSID_CAPACITY,
          Capacity,
```

```
69          VSID_HIGH_MARK,
          HighMarkPercent,
70          VSID_LOW_MARK,
          LowMarkPercent,
71          VSID_FILL_LEVEL,
          FillLevel,
72          VSID_ASSIGNED_BINS,
          AssignedBins,
73          VSID_ARCHIVE_ACTION,
          ActionOpt,
74          VSID_AUTOIMPORT_FLAG,
          AutoImportFlag,
75          VSID_AUTOCHECKIN_FLAG,
          AutoCheckinFlag,
76          VSID_ENDFIELD);
77       if (rc)
78       {
79          vst_print_typecapacity(h);
80       }
81       VS_TypeCapacity_Destroy(h);
82    }
83    return(rc);
84 }
```

Notes

The migration policy options for VSID_HIGH_MARK are no action, operator notification, and automatic migration.

When the number of media in a media type classification reaches the high mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy option is set to notify.
- Initiates automatic migration of media if the migration policy is set to migrate.

When the number of media in a media type classification drops to the low mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy is set to notify.
- Terminates automatic migration of media if the migration policy is set to migrate.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VS_TypeCapacity_Create(l),
- VS_TypeCapacity_Destroy(l),
- VS_TypeCapacity_GetFields(l)

VSCMD_ ArchiveQuery

VSCMD_ArchiveQuery queries for information associated with a VolServ archive.

Upon receipt of an Archive Query command, VolServ obtains information about the specified archive. This information is returned to the client in the status of the command.

Synopsis

VST_BOOLEAN VSCMD_ArchiveQuery
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- handle = The command handle for the Archive Query command.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to use for this command. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive to be queried. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Parameter Type	Description
VSID_ARCHIVE_QUERY_OPTION (VST_ARCHIVE_QUERY_OPTION)	Specifies the type (or types) of archive information being requested. Valid VSID_ARCHIVE_QUERY_OPTION values are defined in the vs_defs.h file. Multiple values can be specified by connecting them with the "|" operator.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is being requested for a single archive or for all archives. Valid VSID_QUERY_OPTION values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software is to wait for final status) and VSE_FALSE (API software is not to wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE).

Parameter Type	Description
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in `USER_FIELD` for this ArchiveQuery command request. `USER_FIELD` is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses `USER_FIELD`.

Return Values

`VSCMD_ArchiveQuery` returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- `VSE_FALSE` - The command failed. A return code of `VSE_FALSE` (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code's value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
    - If the object code's value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
    - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_archivequery_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_ArchiveQuery
       API call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_archivequery_execute(void)
14 #else
15    VST_BOOLEAN
         vst_archivequery_execute()
16 #endif
17 {
18    VST_BOOLEAN                rc =
         VSE_FALSE;
19    int                        ok;
20    VST_QUERY_OPTION           queryopt;
21    VST_ARCHIVE_QUERY_OPTION
         arcqueryopt;
22    VST_ARCHIVE_NAME           archive;
23    VST_COMMAND_HANDLE         cmd;
24
25    /* get parameters from user */
26    printf( "*** Archive Query parameters
         ***\n" );
27    printf("Query all archives? (1) yes,
         (0) no: ");
28    queryopt = (VST_QUERY_OPTION)
         atoi(gets(input));
29
30    if (queryopt == 0)
31    {
```

```
32       printf( "\nEnter Archive Name
         ==>");
33       gets(archive);
34    }
35    else
36    {
37       strcpy(archive, VSD_STRING_INIT);
38    }
39    /* Initialize the Archive Query
         Option */
40    arcqueryopt =
         VSD_ARCQRY_OPTION_NONE;
41    printf( "query drive information? (1)
         yes, (0) no: ");
42    ok = atoi(gets(input));
43    if (ok)
44    {
45       /* Add the option to query for */
46       /* drive information */
47       arcqueryopt +=
         VSD_ARCQRY_OPTION_DRIVE;
48    }
49
50    printf( "query media information? (1)
         yes, (0) no: ");
51    ok = atoi(gets(input));
52    if (ok)
53    {
54       /* add the option to query for
         media */
55       /* information */
56       arcqueryopt +=
         VSD_ARCQRY_OPTION_MEDIA;
57    }
58
59    printf( "query media class
         information? (1) yes, (0) no: ");
60    ok = atoi(gets(input));
61    if (ok)
62    {
63       /* add the option to query media
         class */
```

```
64          /* information */
65          arcqueryopt +=
            VSD_ARCQRY_OPTION_CLASS;
66       }
67
68       printf( “query media type
            information? (1) yes, (0) no: “);
69       ok = atoi(gets(input));
70       if (ok)
71       {
72          /* add the option to query media
            type */
73          /* information */
74          arcqueryopt +=
            VSD_ARCQRY_OPTION_TYPE;
75       }
76       /* create the command handle */
77       /* Note that the command handle is
            not destroyed in */
78       /* this routine, but in vst_dispatch
            when final */
79       /* status is received. */
80       cmd = VS_Command_Create();
81       if ( cmd != (VST_COMMAND_HANDLE)
            NULL)
82       {
83          /* Send the command to the
            VolServ. */
84          /* Note that status is not
            processed here. */
85          /* Instead, it is processed in the
            */
86          /* vst_dispatch routine. Also,
            note that */
87          /* default values such as timeout,
            value */
88          /* retry limit parameters and
            priority are */
89          /* set as default parameters. */
90          if (queryopt ==
            VSE_QUERY_OPTION_ALL)
91          {
```

```
92          /* query all archives */
93          rc = VSCMD_ArchiveQuery(cmd,
94          VSID_QRY_OPTION, queryopt,
95             VSID_ARCHIVE_QRY_OPTION,
        arcqueryopt,
96             VSID_ENDFIELD);
97       }
98       else
99       {
100         /* query a specific archive */
101         rc = VSCMD_ArchiveQuery(cmd,
102            VSID_QRY_OPTION, queryopt,
103            VSID_ARCHIVE_QRY_OPTION,
        arcqueryopt,
104            VSID_ARCHIVE_NAME, archive,
105            VSID_ENDFIELD);
106      }
107   }
108   return ( rc );
109}
```


Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the API cannot receive status for this request.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on command requests submitted through the API interface to the VolServ system.

VolServ generates intermediate status in response to an Archive Query request if:

- Information is being requested for more than one archive.
- Any archive query option is specified.

Archive Query statuses are cumulative. Each status is added to the previous status; therefore, after the final status, the status handle contains all needed information.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for the Archive Query command are set with `VSCMD_ArchiveQuery_SetDefaults`. If command-specific defaults are set for the Archive Query command, they override the global defaults for all Archive Query requests.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Archive Query request, the parameter identifier and the value to be used for the parameter can be submitted on the command request itself.

The following fields can be retrieved from the status handle after a successful Archive Query request:

- `VSID_ARCHIVE_HANDLE`,
- `VSID_ARCHIVE_HANDLE_ENTRY`,
- `VSID_ARCHIVE_HANDLE_TABLE`,
- `VSID_QUERY_OPTION`, `VSID_SEQUENCE_NUM`,

- `VSID_SEQUENCE_TABLE`, `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

`VSCMD_ArchiveQuery` does not trigger any MediaClass callbacks from VolServ.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Archive_GetFields(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Command_GetFields(l),
- VS_Error_GetFields(l),
- VS_Global_GetFields(l),
- VS_Global_SetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VS_Table_GetFields(l),
- VSCMD_ArchiveQuery_SetDefaults(l)

VSCMD_ ArchiveQuery _SetDefaults

`VSCMD_ArchiveQuery_SetDefaults` sets the command-level default parameters for Archive Query command requests.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for the Archive Query command are set with `VSCMD_ArchiveQuery_SetDefaults`. If command-specific defaults are set for the Archive Query command, they override the global defaults for all Archive Query requests.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Archive Query request, the parameter identifier and the value to be used for the parameter can be submitted on the command request itself.

Synopsis

```
VST_BOOLEAN VSCMD_ArchiveQuery_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "..." = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive to be queried on all Archive Query requests. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ARCHIVE_QUERY_OPTION (VST_ARCHIVE_QUERY_OPTION)	Type (or types) of information being requested on Archive Query requests. Valid VSID_ARCHIVE_QUERY_OPTION values are defined in the *vs_defs.h* file. Multiple values may be specified by connecting them with the "|" operator.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on Archive Query requests.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Archive Query requests.
VSID_PRIORITY (VST_PRIORITY)	Execution priority for Archive Query requests. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is being requested for a single specified archive or for all archives. Valid VSID_QUERY_OPTION values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Archive Query requests. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software is to wait for final status from VolServ (or to time-out) for Archive Query requests. Valid options are VSE_TRUE (API software is to wait for final status) and VSE_FALSE (API software is not to wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE).
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for every Archive Query command request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Archive Query requests. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_ArchiveQuery_SetDefaults` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_archivequery_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_ArchiveQuery API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_archivequery_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_archivequery_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
```

Functions

```
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Archive Query default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_ArchiveQuery_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_ArchiveQuery(l)

VSCMD_ ArchiveVary

`VSCMD_ArchiveVary` changes the state of a VolServ configured archive. Name of the archive and the target state must be specified.

Upon receipt of a `VSCMD_ArchiveVary` command, VolServ attempts to change the state of the specified archive. The return code presented to the client indicates the success or failure of the command.

Synopsis

VST_BOOLEAN VSCMD_ArchiveVary
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The command handle for the Archive Vary command.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to apply to this command. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive to be varied. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on this command.
VSID_COMP_STATE (VST_COMP_STATE)	State where the archive is varied. Valid `VSID_COMP_STATE` values are enumerated in the *vs_types.h* file.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Valid options are `VSE_TRUE` (API software is to wait for final status) and `VSE_FALSE` (API software is not to wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`).

Parameter Type	Description
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in the USER_FIELD for this command. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_ArchiveVary returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
    - If the object code's value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
    - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_archivevary_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_ArchiveVary
       API call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_archivevary_execute(void)
14 #else
15    VST_BOOLEAN
         vst_archivevary_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_ARCHIVE_NAME     archive;
20    VST_COMP_STATE       state;
21    VST_COMMAND_HANDLE   cmd;
22
23    /* get parameters from user */
24    printf("*** Archive Vary parameters
         ***\n" );
25    printf("\nEnter Archive ");
26    gets(archive);
27    printf("\nEnter Archive State (1)
         ONLINE (2) OFFLINE (3) DIAG: ");
28    state = (VST_COMP_STATE)
         atoi(gets(input));
29
30    /* create the command handle */
31    /* Note that the command handle is
         not */
32    /* destroyed in this routine, but in
         */
```

```
33    /* vst_dispatch when final status is
         received. */
34    cmd = VS_Command_Create();
35    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
36    {
37       /* Send the command to the
         VolServ. */
38       /* Note that status is not
         processed here. */
39       /* Instead, it is processed in the
         */
40       /* vst_dispatch routine. Also,
         note that */
41       /* default values such as timeout,
         value */
42       /* retry limit and priority are
         set as */
43       /* default parameters.*/
44       rc = VSCMD_ArchiveVary(cmd,
45                      VSID_COMP_STATE,
         state,
46                      VSID_ARCHIVE_NAME,
                  archive,
47                      VSID_ENDFIELD);
48    }
49    return ( rc );
50 }
```


Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the API is not able to receive status for this request.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on command requests submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for the Archive Vary command are set with `VSCMD_ArchiveVary_SetDefaults`. If command-specific defaults are set for the Archive Vary command, they override the global defaults for all Archive Vary requests.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Archive Vary request, the parameter identifier and the value to be used for the parameter can be submitted on the command request itself.

The following fields can be retrieved from the status handle after a successful reprioritize request:

- `VSID_ARCHIVE_NAME`,
- `VSID_COMP_STATE`,
- `VSID_SEQUENCE_NUMBER`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,

- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

VolServ does not generate intermediate status in response to an Archive Vary request.

VSCMD_ArchiveVary does not trigger any MediaClass callbacks from VolServ.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VS_Status_GetFields(l),
- VSCMD_ArchiveVary_SetDefaults(l)

VSCMD_ ArchiveVary_ SetDefaults

`VSCMD_ArchiveVary_SetDefaults` sets the command-level default parameters for Archive Vary command requests.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for the Archive Vary command are set with `VSCMD_ArchiveVary_SetDefaults`. If command-specific defaults are set for the Archive Vary command, they override the global defaults for all Archive Vary requests.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Archive Vary request, the parameter identifier and the value to be used for the parameter can be submitted on the command request itself.

Synopsis

```
VST_BOOLEAN VSCMD_ArchiveVary_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "..." = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive to be varied on Archive Vary requests. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on Archive Vary requests.
VSID_COMP_STATE (VST_COMP_STATE)	State where the archive is varied on Archive Vary requests. Valid VSID_COMP_STATE values are enumerated in the *vs_types.h* file.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on Archive Vary requests.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for Archive Vary requests. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Archive Vary requests. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software is to wait for final status from VolServ (or to timeout) for a command. Valid options are VSE_TRUE (API is to wait for final status) and VSE_FALSE (API is not to wait for final status). Valid options are VSE_TRUE (API software is to wait for final status) and VSE_FALSE (API software is not to wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE).
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for every Archive Vary command request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_ArchiveVary_SetDefaults` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_archivevary_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_ArchiveVary API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_archivevary_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_archivevary_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
```

```
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Archive Vary default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_ArchiveVary_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_ArchiveVary(l)

VSCMD_Audit

`VSCMD_Audit` performs archive inventory verification of the specified archive.

- If the specified archive is robotically controlled, the robot scans each physical bin location and verifies that the database is consistent with the actual location of media. Any detected inconsistencies are returned to the client, logged in a system log file, and VolServ take appropriate action based on the circumstances of the discrepancy.
- On the other hand, if the specified archive is a manual archive, the archive operator is directed to generate the audit report. The operator may then request the report be printed or verify the information on-line. Either way, the operator must actually perform the inventory and then correct any reported discrepancies. Discrepancies are resolved by issuing appropriate media management commands (for example, Move and Eject commands) to relocate media to the reported locations. Inconsistencies detected in a manual archive are not reported to the client.

Audit requests are for full archive audits only; no subset audits are permitted from the API. Subset audits are conducted from the system operator GUI. Full archive audits can be lengthy and must be requested with discretion.

Synopsis

```
VST_BOOLEAN VSCMD_Audit
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)
```

Arguments

- `handle` = Handle of the Audit command.
- "..." =Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The parameter identifiers and types this function accepts are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive to audit. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Client dispatch routine for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software is to wait for final status) and VSE_FALSE (API software is not to wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE).
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this Audit request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Audit returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.

  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code's value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.
- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code's value is `VSE_VOLSERV`, the error occurred in VolServ, and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code's value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API, and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NOTINITIALIZED` - The VolServ API is not initialized.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.
- `VSE_ERR_SEND` - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_audit_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_Audit API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_audit_execute(void)
14 #else
15    VST_BOOLEAN vst_audit_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_ARCHIVE_NAME     archive;
20    VST_COMMAND_HANDLE   cmd;
21
22    /* get parameters from user */
23    printf("*** Audit parameters ***\n"
         );
24    printf("Enter Archive Name ==> " );
25    gets(archive);
26
27    /* create the command handle */
28    /* Note that the command handle is
         not */
29    /* destroyed in this routine, but in
         */
30    /* vst_dispatch when finalstatus is
         received. */
31    cmd = VS_Command_Create();
32    if (cmd != (VST_COMMAND_HANDLE )NULL)
33    {
34       /* Send the command to the VolServ
         software. */
```

```
35       /* Note that status is not
         processed here. */
36       /* Instead, it is processed in the
         */
37       /* vst_dispatch routine. Also,
         note that */
38       /* default values such as timeout,
         value */
39       /* retry limit and priority are
         set as*/
40       /* default parameters. */
41       rc = VSCMD_Audit(cmd,
42                        VSID_ARCHIVE_NAME,
         archive,
43                        VSID_ENDFIELD);
44    }
45    return ( rc );
46 }
```


Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`. Because of the time required for robotic audits, the time-out value or retries may need to be increased from the API default values.

VolServ can generate intermediate status in response to an Audit request.

`VSCMD_Audit` can trigger MediaClass callbacks from VolServ.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the API is not able to receive status for this request.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on command requests submitted through the API interface to the VolServ system. With the exceptions of the manual archives, a pending or executing `Audit` command can be cancelled using the VolServ `Cancel` command.

A pending or executing `Audit` command can be reprioritized using the VolServ `Reprioritize` command.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for the Audit command are set with `VSCMD_Audit_SetDefaults`. If command-specific defaults are set for the Audit command, they override the global defaults for all Audit requests.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Audit request, the parameter identifier and the value to be used for the parameter can be submitted on the command request itself.

The following fields can be retrieved from the status handle after a successful Audit:

- `VSID_ACTION_CODE`,
- `VSID_ACTION_CODE_ENTRY`,
- `VSID_ACTION_CODE_TABLE`,
- `VSID_COMP_ID`,

- VSID_COMP_ID_ENTRY,
- VSID_COMP_ID_TABLE,
- VSID_ERROR,
- VSID_ERROR_ENTRY,
- VSID_ERROR_TABLE,
- VSID_MEDIA_ID,
- VSID_MEDIA_ID_ENTRY,
- VSID_MEDIA_ID_TABLE,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD,
- VSID_WAIT_REASON.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Command_GetFields(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VS_Initialize(l),
- VS_Media_GetFields(l),
- VS_Status_GetFields(l),
- VS_Table_GetFields(l),
- VSCMD_Audit_SetDefaults(l)

VSCMD_Audit_SetDefaults

`VSCMD_Audit_SetDefaults` sets command-level default parameters for Audit commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Audit commands are set with `VSCMD_Audit_SetDefaults`. If command-specific defaults are set for Audit commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Audit command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_Audit_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- **“…”** = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive to audit. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive intermediate and final status for Audit commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status for Audit commands.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for Audit commands.Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Audit commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Audit commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to be put in USER_FIELD for Audit commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Audit commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Audit_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_audit_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Audit API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN vst_audit_defaults(void)
15 #else
16    VST_BOOLEAN vst_audit_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Audit default parameters
         ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
```

```
31    rc = VSCMD_Audit_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Audit(l)

VSCMD_ Cancel

`VSCMD_Cancel` stops a pending request from executing or aborts an executing Audit request.

Upon receipt of a Cancel request, VolServ returns final status to the client that requested the cancellation. If the specified request is capable of being cancelled, a good status is returned. Otherwise, a status that indicates why the request cannot be cancelled is returned.

If the specified request is pending execution, the request is directed to terminate after any allocated resources are released. A final status that indicates “failure due to cancellation” is sent to the original issuer of the cancelled request.

Upon receipt of a Cancel request of an executing Audit request, the processing is directed to abort, and the remainder of the Audit request is cancelled. A final status that indicates “failure due to cancellation” is sent to the original issuer of the cancelled request.

To issue a valid Cancel request, the client must supply the request identifier and request type of the request to be canceled. The `VSCMD_Cancel` function can either take these parameters separately or retrieve them from a command handle.

Synopsis

```
VST_BOOLEAN VSCMD_Cancel
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)
```

Arguments

- `handle` = The command handle for this Cancel request.
- `"..."` = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_COMMAND_HANDLE (VST_COMMAND_HANDLE)	The command handle of the request to cancel.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_REQUEST_ID (VST_REQUEST_ID)	The request identifier of the request to cancel.
VSID_REQUEST_TYPE (VST_REQUEST_TYPE)	The request type of the request to cancel. Valid `VSID_REQUEST_TYPE` values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_Cancel` returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- `VSE_FALSE` - The request failed. A return code of `VSE_FALSE` (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.
- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code value is `VSE_VOLSERV`, the error occurred in VolServ and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.

- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_cancel_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_Cancel API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_cancel_execute(void)
14 #else
15    VST_BOOLEAN vst_cancel_execute()
16 #endif
17 {
18    VST_BOOLEAN            rc =
         VSE_FALSE;
19    VST_REQUEST_ID         req;
20    VST_REQUEST_TYPE       c;
21    VST_COMMAND_HANDLE     cmd;
```

```
22
23    /* get parameters from user */
24    printf("*** Cancel parameters ***\n"
         );
25    printf("Enter Request ID ==> " );
26    req = (VST_REQUEST_ID)
         atol(gets(input));
27    printf("Enter Command Request Type
         ==> " );
28    c = (VST_REQUEST_TYPE)
         atol(gets(input));
29
30    /* create the command handle */
31    /* Note that the command handle is
         not */
32    /* destroyed in this routine, but in
         */
33    /* vst_dispatch when final status is
         received. */
34    cmd = VS_Command_Create();
35    if (cmd != (VST_COMMAND_HANDLE )NULL)
36    {
37       /* Send the command to the VolServ
         software. */
38       /* Note that status is not
         processed here. */
39       /* Instead, it is processed in the
         */
40       /* vst_dispatch routine. Also,
         note that */
41       /* default values such as timeout,
         value */
42       /* retry limit and priority are
         set as */
43       /* default parameters. */
44       rc = VSCMD_Cancel(cmd,
45
         VSID_REQUEST_ID, req,
46
         VSID_REQUEST_TYPE, c,
47                          VSID_ENDFIELD);
48    }
```

```
49    return ( rc );
50 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Cancel request.

`VSCMD_Cancel` does not trigger any MediaClass callbacks from VolServ.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Cancel request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.

- Command-specific parameter defaults for Cancel commands are set with `VSCMD_Cancel_SetDefaults`. If command-specific defaults are set for Cancel commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Cancel command, the parameter identifier and the value to be used for the parameter can be submitted on the specific command itself.

The following fields can be retrieved from the status handle after a successful Cancel request:

- `VSID_REQUEST_ID`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_Cancel_SetDefaults(l)

VSCMD_ Cancel_ SetDefaults

`VSCMD_Cancel_SetDefaults` sets command-level default parameters for Cancel commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Cancel commands are set with `VSCMD_Cancel_SetDefaults`. If command-specific defaults are set for Cancel commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Cancel command, the parameter identifier and the value to be used for the parameter can be submitted on the specific command itself.

Synopsis

```
VST_BOOLEAN VSCMD_Cancel_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "..." = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for Cancel commands.
VSID_COMMAND_HANDLE (VST_COMMAND_HANDLE)	The command handle of the request to cancel.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on Cancel commands.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for Cancel commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_REQUEST_ID (VST_REQUEST_ID)	The request identifier of the request to cancel.
VSID_REQUEST_TYPE (VST_REQUEST_TYPE)	The request type of the request to cancel. Valid VSID_REQUEST_TYPE values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Cancel commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Indicates whether the API software waits for final status from VolServ (or times-out) for Cancel commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The total wait time for a command is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to be put in USER_FIELD for Cancel commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Cancel commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_Cancel_SetDefaults` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_cancel_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Cancel API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_cancel_defaults(void)
15 #else
16    VST_BOOLEAN vst_cancel_defaults()
17 #endif
18 {
19    VST_BOOLEAN          rc = VSE_FALSE;
20    VST_PRIORITY         priority;
```

```
21    VST_USER_FIELD       user_field;
22    VST_TIME_OUT         timeout;
23    VST_RETRY_LIMIT      retries;
24    VST_STATUS_WAIT_FLAG wait_flag;
25    VST_ENTERPRISE_ID    enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Cancel default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_Cancel_SetDefaults(
32          VSID_PRIORITY,
         priority,
33          VSID_USER_FIELD,
         user_field,
34          VSID_TIMEOUT_VALUE,
         timeout,
35          VSID_RETRY_LIMIT,
         retries,
36          VSID_STATUS_WAIT_FLAG,
         wait_flag,
37          VSID_ENTERPRISE_ID,
         enterprise_id,
38          VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Cancel(l)

VSCMD_ Checkin

VSCMD_Checkin checks media into the VolServ system. Only media that have been previously checked out can be checked in.

Checkin is a logical operation. After a medium is logically checked into the VolServ system, the medium must be physically entered into an archive before becoming available for client use (mounting,…).

A medium is physically entered into the VolServ system via the Enter functionality available from the appropriate archive's console display. The Enter functionality is not available through the API interface.

If a destination archive is not specified on a VSCMD_Checkin request, the media are returned to the archive where they were checked-out.

Synopsis

```
VST_BOOLEAN VSCMD_Checkin
(VST_COMMAND_HANDLE
handle,
"…",
VSID_ENDFIELD)
```

Arguments

- handle = The command handle for this Checkin request.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive where the specified media are to be checked-in. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_MEDIA_ID_LIST (int)	Number of media to check-in.
(char **)	An array of the identifiers of the media to check-in.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_CheckIn returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The request failed.A return code of VSE_FALSE (which is 0) means the request failed.

  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.

- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code value is `VSE_VOLSERV`, the error occurred in VolServ, and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API, and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NOTINITIALIZED` - The VolServ API is not initialized.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_checkin_execute
4  *
5  * PURPOSE:
6  * This function sends a checkin command
       to the
7  * VolServ software, prompting for all
       values needed.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_checkin_execute(void)
15 #else
16    VST_BOOLEAN vst_checkin_execute()
17 #endif
18 {
19    VST_BOOLEAN          rc = VSE_FALSE;
20    int                  count;
21    char                 *
         medialist[VST_MAX_ITEMS];
22    VST_ARCHIVE_NAME     archive;
23    VST_COMMAND_HANDLE   cmd;
24
25    /* get parameters from user */
26    printf("*** Check-in parameters
         ***\n" );
27    printf("\nEnter Archive name (return
         for default) ");
```

```
28    gets( archive);
29    count = vst_getmedialist(medialist,
         VST_MAX_ITEMS);
30    /* create the command handle Note
         that the */
31    /* command handle is not destroyed in
         this*/
32    /* routine, but in vst_dispatch when
         final */
33    /* status is received. */
34    cmd = VS_Command_Create();
35    /* validate the command handle */
36    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
37    {
38       /* Send the command to the VolServ
         software. */
39       /* Note that status is not
         processed here. */
40       /* Instead, it is processed in the
         */
41       /* vst_dispatch routine. *Also,
         note that */
42       /* default values such as timeout,
         value */
43       /* retry limit and priority are
         set as */
44       /* default parameters. */
45       rc = VSCMD_Checkin(cmd,
46                   VSID_ARCHIVE_NAME,
         archive,
47                   VSID_MEDIA_ID_LIST,
         count, medialist,
48                   VSID_ENDFIELD);
49    }
50
51    return ( rc );
52 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function may be executed.

VolServ does not generate intermediate status messages for a Checkin request.

`VSCMD_Checkin` can trigger MediaClass callbacks from VolServ.

Media must be checked-out before they can be specified on a Checkin request.

Media checked-out of one archive can be checked into another archive, as long as the MediaClass group where the media belong are associated with archive media class(es) in the receiving archive.

Failure of a Checkin request for one medium does not fail the request for all specified media.

Media checked out from more than one archive can be checked in as a single group into a single new archive (assuming necessary archive media class associations are defined.)

Media that are checked out from more than one archive and are checked in as a single group without a target archive specified (the archive name is set to the empty string, " ") are returned to their respective check-out archives.

The total length of time the API software waits for a command status from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on a Checkin request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Checkin commands are set with `VSCMD_Checkin_SetDefaults`. If command-specific defaults are set for Checkin commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Checkin command, the parameter identifier and the value to be used for the parameter can be submitted on the specific command itself.

The following fields can be retrieved from the status handle after a successful Checkin command:

- `VSID_ERROR_CODE`,
- `VSID_ERROR_CODE_ENTRY`, `VSID_ERROR_CODE_TABLE`,
- `VSID_MEDIA_ID`, `V`
- `SID_MEDIA_ID_ENTRY`,
- `VSID_MEDIA_ID_TABLE`,

- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_Checkin_SetDefaults(l)

VSCMD_ Checkin_ SetDefaults

`VSCMD_Checkin_SetDefaults` sets command-level default parameters for Checkin commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Checkin commands are set with `VSCMD_Checkin_SetDefaults`. If command-specific defaults are set for Checkin commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Checkin command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_Checkin_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- “…” = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive inwhere the specified media are to be checked-in. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive intermediate and final status for Checkin commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status for Checkin commands.
VSID_MEDIA_ID_LIST (int)	Number of media to check-in.
(char **)	An array of the identifiers of the media to check-in.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for Checkin commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Checkin commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Checkin commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The total wait time for a command is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Checkin commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this Checkin command. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Checkin_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_checkin_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Checkin API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       **********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_checkin_defaults(void)
15 #else
16    VST_BOOLEAN vst_checkin_defaults()
17 #endif
18 {
19    VST_BOOLEAN          rc = VSE_FALSE;
20    int                  count;
21    VST_PRIORITY         priority;
22    VST_USER_FIELD       user_field;
23    VST_TIME_OUT         timeout;
24    VST_RETRY_LIMIT      retries;
25    VST_STATUS_WAIT_FLAG wait_flag;
26    VST_ENTERPRISE_ID    enterprise_id;
27    char
         archive[VSD_ARCHIVE_NAME_LEN];
28
29    /* get parameters from user */
30    printf("*** Check-in default
         parameters ***\n" );
```

```
31    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
32    printf("\nEnter Archive ");
33    gets( archive);
34    /* set the default parameters */
35    rc = VSCMD_Checkin_SetDefaults(
36          VSID_PRIORITY,
         priority,
37          VSID_USER_FIELD,
         user_field,
38          VSID_TIMEOUT_VALUE,
         timeout,
39          VSID_RETRY_LIMIT,
         retries,
40          VSID_STATUS_WAIT_FLAG,
         wait_flag,
41          VSID_ENTERPRISE_ID,
         enterprise_id,
42          VSID_ARCHIVE_NAME,
         archive,
43          VSID_ENDFIELD);
44
45    return ( rc );
46 }
```


Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Checkin(l)

VSCMD_ Checkout

VSCMD_Checkout checks media out of the VolServ system. Media that are checked out are still known by VolServ, but are unavailable for client allocation.

Upon receipt of a Checkout request, VolServ marks the specified media for check out. If the specified media are contained in archives, VolServ adds the media to the eject candidate list of the containing archive. An operator must select the Eject functionality from the appropriate archive's console display to physically remove the checked-out media from the containing archive. The VolServ Eject functionality is not available over the API interface.

The client may specify a comment on the VSCMD_Checkout command. This comment is displayed on the applicable archive console eject list for each medium being checked out.

Synopsis

```
VST_BOOLEAN VSCMD_Checkout
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)
```

Arguments

- handle = The command handle for this Checkout request.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_COMMENT (VST_COMMENT)	The text information (comment) to appear on the appropriate archive console eject list for each medium specified for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_MEDIA_ID_LIST (int)	Number of media to check-out.
(char **)	An array of the identifiers of the media to check-out.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are `VSE_TRUE` (API software waits for final status) and `VSE_FALSE` (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE.`

Parameter Type	Description
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Checkout returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
    - If the object code value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
    - If the object code value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_checkout_execute
4  *
5  * PURPOSE:
6  * This function sends a checkout command
       to the
7  * VolServ software, prompting for all
       values needed.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_checkout_execute(void)
15 #else
16    VST_BOOLEAN vst_checkout_execute()
17 #endif
18 {
19    VST_BOOLEAN          rc = VSE_FALSE;
20    int                  count;
21    char                 *
         medialist[VST_MAX_ITEMS];
22    VST_COMMENT          comment;
23    VST_COMMAND_HANDLE   cmd;
24
25    /* get parameters from user */
26    printf("*** Check-out Parameters
         ***\n" );
27    printf("\nEnter Checkout Comment ");
28    gets( comment);
29    count = vst_getmedialist(medialist,
         VST_MAX_ITEMS);
30    /* create the command handle */
31    /* Note that the command handle is
         not */
32    /* destroyed in this routine, but in
         */
```

```
33    /* vst_dispatch when finalstatus is
         received. */
34    cmd = VS_Command_Create();
35    /* validate the command handle */
36    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
37    {
38          /* Send the command to the
         VolServ */
39          /* software. Note that status
         is not */
40          /* processed here. Instead, it
         is */
41          /* processed it the
         vst_dispatch */
42          /* routine. Also, note that
         default */
43          /* values such as timeout,
         value retry */
44          /* limit and priority are set
         as default */
45          /* parameters. */
46       rc = VSCMD_Checkout(cmd,
47             VSID_COMMENT, comment,
48             VSID_MEDIA_ID_LIST, count,
         medialist,
49             VSID_ENDFIELD);
50    }
51
52    return ( rc );
53 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ does not generate intermediate status messages in response to a Checkout request.

A Checkout request can trigger unsolicited status messages from VolServ.

Checked-out media are unavailable for use by VolServ clients.

Failure of a Checkout request for one medium does not fail the request for all specified media.

A medium can be checked out even if it is currently allocated. Attempts to physically eject an allocated medium fail until the medium is no longer is use.

The Checkout request only marks a specified medium for logical check out and places the medium on the appropriate archive's Eject list. The medium is physically removed from the archive when the operator ejects the medium from the archive.

A medium marked for check out can be unmarked (removed from the Eject list) by the ClearEject request. An operator can also remove a medium from the Eject list by performing an Eject Fail operation from an archive console. The Eject Fail functionality is available over the API interface via the ClearEject request.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on a Checkout request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Checkin commands are set with `VSCMD_Checkout_SetDefaults`. If command-specific defaults are set for Checkout commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Checkout command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Checkout request:

- `VSID_ERROR_CODE`,
- `VSID_ERROR_CODE_ENTRY`,
- `VSID_ERROR_CODE_TABLE`,
- `VSID_MEDIA_ID`,
- `VSID_MEDIA_ID_ENTRY`,
- `VSID_MEDIA_ID_TABLE`,

- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_Checkout_SetDefaults(l

VSCMD_ Checkout_Set Defaults

`VSCMD_Checkout_SetDefaults` sets the command-level default parameters for Checkout commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Checkout commands are set with `VSCMD_Checkout_SetDefaults`. If command-specific defaults are set for Checkout commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Checkout command, the parameter identifier and the value to be used for the parameter can be submitted on the specific command itself.

Synopsis

```
VST_BOOLEAN VSCMD_Checkout_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- “…” = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for Checkout commands.
VSID_COMMENT (VST_COMMENT)	The text information (comment) to appear on the appropriate archive console Eject list for each medium specified for Checkout commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status for Checkout commands.
VSID_MEDIA_ID_LIST (int)	Number of media to check out with Checkout commands.
(char **)	An array of the identifiers of the media to check out with Checkout commands.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for Checkout commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Checkout commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Checkout commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Checkout commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Checkout commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Checkout_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_checkout_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Checkout API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_checkout_defaults(void)
15 #else
16    VST_BOOLEAN vst_checkout_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26    VST_COMMENT             comment;
27
28    /* get parameters from user */
29    printf("*** Check-out Default
         Parameters ***\n" );
```

```
30    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
31    printf("\nEnter Checkout Comment ");
32    gets( comment);
33    /* set the default parameters */
34    rc = VSCMD_Checkout_SetDefaults(
35          VSID_PRIORITY,
         priority,
36          VSID_USER_FIELD,
         user_field,
37          VSID_TIMEOUT_VALUE,
         timeout,
38          VSID_RETRY_LIMIT,
         retries,
39          VSID_STATUS_WAIT_FLAG,
         wait_flag,
40          VSID_ENTERPRISE_ID,
         enterprise_id,
41          VSID_COMMENT,
         comment,
42          VSID_ENDFIELD);
43
44    return ( rc );
45 }
```


Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Checkout(l)

VSCMD_ClearEject

`VSCMD_ClearEject` reverses the scheduled ejection of one or more media from an archive.

Ejects can be generated during processing of the VolServ Checkout, Export, Mount, and Move commands. Ejects can also be generated during automatic migration of media.

The ClearEject command essentially undoes the completion of these commands. Media are removed from the Eject list and returned to the available state.

For example, if a client issues an Export request for a specific medium, the specified medium is scheduled for removal by adding the medium to the Eject list for the archive associated with the medium. If the client decides the medium should not be removed from its associated archive, the client can issue a ClearEject request, and VolServ removes the medium from the Eject list.

Synopsis

```
VST_BOOLEAN VSCMD_ClearEject
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)
```

Arguments

- `handle` = The command handle for this ClearEject request.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for ClearEject commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status for ClearEject commands.
VSID_MEDIA_ID_LIST (int)	Number of media to remove from the Eject list.
(char **)	An array of the identifiers of the media to remove from the Eject list.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for ClearEject commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software ClearEject commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for ClearEject commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The total wait time for a command is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for ClearEject commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for ClearEject commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_ClearEject returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.

- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_clearejct_execute
4  *
5  * PURPOSE:
6  * This function sends an clearejct
       command to the
7  * VolServ software, prompting for all
       values needed.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_clearejct_execute(void)
15 #else
16    VST_BOOLEAN vst_clearejct_execute()
17 #endif
18 {
19    VST_BOOLEAN          rc = VSE_FALSE;
20    int                  count;
21    char                 *
         medialist[VST_MAX_ITEMS];
22    VST_COMMAND_HANDLE   cmd;
23
24    /* get parameters from user */
25    printf("*** ClearEject Parameters
         ***\n" );
26    count = vst_getmedialist(medialist,
         VST_MAX_ITEMS);
27    /* create the command handle */
```

```
28    /* Note that the command handle is
         not */
29    /* destroyed in this routine, but in
         */
30    /* vst_dispatch when final status is
         received. */
31    cmd = VS_Command_Create();
32    /* validate the command handle */
33    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
34    {
35       /* Send the command to the VolServ
         software. */
36       /* Note that status is not
         processed here. */
37       /* Instead, it is processed in the
         */
38       /* vst_dispatch routine. Also,
         note that */
39       /* default values such as timeout,
         value */
40       /* retry limit and priority are
         set as */
41       /* default parameters. */
42       rc = VSCMD_ClearEject(cmd,
43          VSID_MEDIA_ID_LIST, count,
         medialist,
44          VSID_ENDFIELD);
45    }
46
47    return ( rc );
48 }
```


Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ does not generate intermediate status messages in response to a ClearEject request.

A ClearEject request can trigger MediaClass callbacks from VolServ.

Failure of a ClearEject request for one medium does not fail the request for all media.

An operator can also remove a medium from the Eject list by performing an Eject Fail from the appropriate archive console. The Eject Fail functionality is not available over the API interface.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on a ClearEject request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for ClearEject commands are set with `VSCMD_ClearEject_SetDefaults`. If

command-specific defaults are set for ClearEject commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a ClearEject command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Checkout request:

- `VSID_ERROR_CODE`,
- `VSID_ERROR_CODE_ENTRY`,
- `VSID_ERROR_CODE_TABLE`,
- `VSID_MEDIA_ID`,
- `VSID_MEDIA_ID_ENTRY`,
- `VSID_MEDIA_ID_TABLE`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_ClearEject_SetDefaults(l)

VSCMD_ClearEject_SetDefaults

`VSCMD_ClearEject_SetDefaults` sets command-level default parameters for ClearEject commands.

Two levels of default parameter settings are used in the API software— global defaults and command-specific parameter defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for ClearEject commands are set with `VSCMD_ClearEject_SetDefaults`. If command-specific defaults are set for the ClearEject command, they override the global defaults for all ClearEject commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a ClearEject command, the parameter identifier and the value to be used for the parameter can be submitted on the specific command itself.

Synopsis

```
VST_BOOLEAN VSCMD_ClearEject_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "..." = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for ClearEject commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status for ClearEject commands.
VSID_MEDIA_ID_LIST (int)	Number of media to remove from the Eject list with ClearEject commands.
(char **)	An array of the identifiers of the media to remove from the Eject list with ClearEject commands.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for ClearEject commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for ClearEject commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for ClearEject commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for ClearEject commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for ClearEject commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_ClearEject_SetDefaults` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_cleareject_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_ClearEject API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_cleareject_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_cleareject_defaults()
17 #endif
18 {
```

```
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** ClearEject Default
         Parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_ClearEject_SetDefaults(
32       VSID_PRIORITY,          priority,
33       VSID_USER_FIELD,
         user_field,
34       VSID_TIMEOUT_VALUE,     timeout,
35       VSID_RETRY_LIMIT,       retries,
36       VSID_STATUS_WAIT_FLAG,
         wait_flag,
37       VSID_ENTERPRISE_ID,
         enterprise_id,
38       VSID_ENDFIELD);
39
40    return ( rc );
41 }
```


Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_ClearEject(l)

VSCMD_ Connect

`VSCMD_Connect` associates a specified Internet address with a specified enterprise identifier. After this association is established, the client can listen for MediaClass callbacks and VolServ statuses that are generated for that enterprise.

Any Internet address can be associated with more than one enterprise identifier. In addition, there is no limit to the number of Internet addresses that can be associated with any given enterprise identifier.

When sending intermediate or final status to the client associated with an enterprise, VolServ uses the enterprise identifier to determine the address to use for the status. This is done by matching the enterprise identifier passed in the request with those in the internal table and using the associated address information.

If an enterprise has multiple clients, each client is kept on a receiving queue. If the first enterprise client cannot receive status or MediaClass callback (because of an RPC error), the next client in the queue is tried. This continues until a client successfully receives the status/callback or until the receiving queue is exhausted. If no client successfully receives the status, it is logged and VolServ invokes its retry scheme. If no client successfully receives a MediaClass callback, it is logged and discarded.

There are two methods for scheduling status/callbacks to enterprise clients:

- Round Robin

The statuses and callbacks are distributed evenly among the connected clients. After a status or callback is received by a client, that client is placed at the end of the receiving queue. This method is used by distributed processing systems that want to distribute the work among clients.

- First Received/First Scheduled

The statuses and callbacks are issued to the client that successfully received the previous status or callback. The only time a different client is tried is when the first client fails (at which time the failed client is placed at the end of the queue).

The scheduling method selected by VolServ is dictated by the `ENTERPRISE_ROUND_SCHEDULING` environmental variable. If this variable is set to ‘Y’, the Round Robin scheduling method is used. Otherwise, the First Received/First Scheduled method is used. (The `ENTERPRISE_ROUND_SCHEDULING` environmental variable is in the *envvar.config* configuration file.)

Synopsis

```
VST_BOOLEAN VSCMD_Connect
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)
```

Arguments

- `handle` = The command handle for this Connect request.
- `“…”` = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value of the field to use for this command. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_CONNECT_HANDLE (VST_CONNECT_HANDLE)	The connect handle that contains the enterprise callback address information to be used by VolServ when returning status and MediaClass callbacks to an enterprise. VSID_CONNECT_HANDLE is not applicable if VSID_PROCEDURE_NUMBER, VSID_PROGRAM_NUMBER, VSID_PROTOCOL, and VSID_VERSION_NUMBER are specified.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	RPC procedure number of the client process to receive status messages and MediaClass callbacks from VolServ. VSID_PROCEDURE_NUMBER is not applicable if VSID_CONNECT_HANDLE is specified.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	RPC program number of the client process to receive status messages and MediaClass callbacks from VolServ. VSID_PROGRAM_NUMBER is not applicable if VSID_CONNECT_HANDLE is specified.

Parameter Type	Description
VSID_PROTOCOL (VST_PROTOCOL)	Internet protocol VolServ uses to return status messages and MediaClass callbacks to this client. The default `VSID_PROTOCOL` is `VSE_PROT_UDP`. `VSID_PROTOCOL` is not applicable if `VSID_CONNECT_HANDLE` is specified.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_SOCKADDR_IN (VST_SOCKADDR_IN)	Internet socket address for this client.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are `VSE_TRUE` (API software waits for final status) and `VSE_FALSE` (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE.`
VSID_TARGET_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The enterprise identifier of the enterprise with which a connection is desired.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	RPC version number of the client process to receive status messages and MediaClass callbacks from VolServ. VSID_VERSION_NUMBER is not applicable if VSID_CONNECT_HANDLE is specified.

Return Values

VSCMD_Connect returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_connect_execute
4  *
5  * PURPOSE:
```

```
6  * This executes the VSCMD_Connect API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
       vst_connect_execute(void)
14 #else
15    VST_BOOLEAN vst_connect_execute()
16 #endif
17 {
18    VST_BOOLEAN             rc =
       VSE_FALSE;
19    VST_ENTERPRISE_ID
       TargetEnterpriseID;
20    VST_SOCKADDR_IN
       socketaddress;
21    VST_PROGRAM_NUMBER      prognum;
22    VST_COMMAND_HANDLE      cmd;
23    VST_VERSION_NUMBER      versnum;
24    VST_PROCEDURE_NUMBER    procnum;
25    int                     temp;
26
27    /* get parameters from user */
28    printf("*** Connect parameters
       ***\n" );
29    printf("Enter Enterprise ID ==> " );
30    TargetEnterpriseID =
       (VST_ENTERPRISE_ID)
       atoi(gets(input));
31    printf("Enter program number ==>");
32    prognum = (VST_PROGRAM_NUMBER)
       atol(gets(input));
33    printf("Enter Version Number ==> " );
34    versnum = (VST_VERSION_NUMBER)
       atol(gets(input));
35    printf("Enter Procedure Number ==> "
       );
```

```
36    procnum = (VST_PROCEDURE_NUMBER)
         atoi(gets(input));
37    printf("Enter Socket sin family ==> "
         );
38    temp = atoi(gets(input));
39    socketaddress.sin_family = (short)
         temp;
40    printf("Enter Socket sin port ==> "
         );
41    temp = atoi(gets(input));
42    socketaddress.sin_port = (u_short)
         temp;
43    printf("Enter Socket sin address ==>
         " );
44    temp = atoi(gets(input));
45    socketaddress.sin_addr = (u_long)
         temp;
46
47    /* create the command handle */
48    /* Note that the command handle is
         not */
49    /* destroyed in this routine, but in
         */
50    /* vst_dispatch when final status is
         received. */
51    cmd = VS_Command_Create();
52    if (cmd != (VST_COMMAND_HANDLE )NULL)
53    {
54       /* Send the command to the VolServ
         software. */
55       /* Note that status is not
         processed here. */
56       /* Instead, it is processed in the
         */
57       /* vst_dispatch routine. Also,
         note that */
58       /* default values such as timeout,
         value */
59       /* retry limit and priority are
         set as */
60       /* default parameters. */
61       rc = VSCMD_Connect(cmd,
```

```
62
        VSID_TARGET_ENTERPRISE_ID,TargetE
        nterpriseID
63       VSID_PROGRAM_NUMBER,    prognum,
64       VSID_VERSION_NUMBER,    versnum,
65       VSID_PROCEDURE_NUMBER,  procnum,
66       VSID_PROTOCOL,
        VSE_PROT_TCP,
67       VSID_SOCKADDR_IN,
        socketaddress,
68       VSID_ENDFIELD);
69    }
70  return ( rc );
71 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Connect command.

The Connect command cannot trigger MediaClass callbacks from VolServ.

The `VSID_CONNECT_HANDLE` parameter may be used after a Connect Query command to reconnect an enterprise after the client has gone down.

The Connect command cannot be cancelled. The client may remove an enterprise/address association by issuing a `VSCMD_Disconnect` command to the VolServ system.

A client may use the `VSCMD_ConnectQuery` command to determine if an enterprise is already registered and, if so, under what address.

If a client sends a `VSCMD_Connect` command for an enterprise that already exists, future status messages may be returned to a different client. When a client specifies "enterprise" in the client header message, the resultant status messages may be returned to any client associated with the same enterprise identifier.

The MediaClass callback receiving queue is kept separate from the command status receiving queue. Each of these queues is scheduled separately. Therefore, a client for an enterprise that receives both statuses and callbacks may receive a command status and a MediaClass callback before another client receives either message.

The `VSCMD_Connect` command can be issued only through the client interface. The association between an enterprise and its client cannot be established via the GUI.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, final status for this command is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Connect command submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

A client can issue the Connect Query command to determine if an enterprise is already registered and, if so, under what address.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for the Connect command are set with `VSCMD_Connect_SetDefaults`. If command-specific defaults are set for the Connect command, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Connect command, the parameter identifier and the value to be used for the parameter can be submitted on the specific command itself.

The following fields can be retrieved from the status handle after a successful Connect command:

- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_TARGET_ENTERPRISE_ID`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Connect_Create(l),
- VS_Connect_Destroy(l),
- VS_Connect_GetFields(l),
- VS_Connect_SetFields(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_ConnectQuery(l),
- VSCMD_Connect_SetDefaults(l)

VSCMD_ Connect_Set Defaults

`VSCMD_Connect_SetDefaults` sets command-level default parameters for Connect commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Create Archive Media Class commands are set with `VSCMD_CreateArchiveMediaClass_SetDefaults`. If command-specific defaults are set for Create Archive Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Create Archive Media Class command, the parameter identifier and the value to be used for

Synopsis

```
VST_BOOLEAN VSCMD_Connect_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- **“…”** = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for Connect commands.
VSID_CONNECT_HANDLE (VST_CONNECT_HANDLE)	The connect handle that contains the enterprise callback address information to be used by VolServ when returning status and MediaClass callbacks to an enterprise. VSID_CONNECT_HANDLE is not applicable if VSID_PROCEDURE_NUMBER, VSID_PROGRAM_NUMBER, VSID_PROTOCOL, and VSID_VERSION_NUMBER are specified.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for Connect requests.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for Connect commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	RPC procedure number of the client process to receive status messages and MediaClass callbacks from VolServ. VSID_PROCEDURE_NUMBER is not applicable if VSID_CONNECT_HANDLE is specified.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	RPC program number of the client process to receive status messages and MediaClass callbacks from VolServ. VSID_PROGRAM_NUMBER is not applicable if VSID_CONNECT_HANDLE is specified.

Parameter Type	Description
VSID_PROTOCOL (VST_PROTOCOL)	Internet protocol VolServ uses to return status messages and MediaClass callbacks to this client. The default VSID_PROTOCOL is VSE_PROT_TCP. VSID_PROTOCOL is not applicable if VSID_CONNECT_HANDLE is specified.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Connect commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_SOCKADDR_IN (VST_SOCKADDR_IN)	Internet socket address for this client.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Connect commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The enterprise identifier of the enterprise with which a connection is desired.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Connect commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Connect commands. Neither the API software nor VolServ uses USER_FIELD.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	RPC version number of the client process to receive status messages and MediaClass callbacks from VolServ. VSID_VERSION_NUMBER is not applicable if VSID_CONNECT_HANDLE is specified.

Return Values

VSCMD_Connect_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_connect_defaults
4  *
5  * PURPOSE:
```

```
6  * This function sets the default
       parameters for the
7  * VSCMD_Connect API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 ************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
       vst_connect_defaults(void)
15 #else
16    VST_BOOLEAN vst_connect_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
       VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
       enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Connect default
       parameters ***\n" );
29    vst_promptforglobals(&priority,
       user_field, &timeout, &retries,
       &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_Connect_SetDefaults(
32          VSID_PRIORITY,
       priority,
33          VSID_USER_FIELD,
       user_field,
34          VSID_TIMEOUT_VALUE,
       timeout,
35          VSID_RETRY_LIMIT,
       retries,
```

```
36              VSID_STATUS_WAIT_FLAG,
          wait_flag,
37              VSID_ENTERPRISE_ID,
          enterprise_id,
38              VSID_ENDFIELD);
39     return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Connect(l)

VSCMD_ Connect-Query

VSCMD_ConnectQuery returns a list of all client Internet addresses currently associated with the specified enterprise identifier.

The Connect Query command can be issued through either the client interface or the GUI. However, only from the GUI can "query all" be specified to list all enterprises. From the client interface, only one enterprise can be specified within a single command. This restriction prevents any client from listing the clients of other enterprises being serviced by VolServ.

Synopsis

```
VST_BOOLEAN VSCMD_ConnectQuery
(VST_COMMAND_HANDLE handle
"…",
VSID_ENDFIELD)
```

Arguments

- handle = The command handle for this Connect Query command.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_ENTERPRISE_ID (VST_REQUEST_ID)	Identifier of the enterprise to be queried.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this command. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are `VSE_TRUE` (API software waits for final status) and `VSE_FALSE` (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE.`
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_ConnectQuery returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

  - If the object code value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.

- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_connectquery_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_ConnectQuery
       API call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_connectquery_execute(void)
14 #else
15    VST_BOOLEAN
         vst_connectquery_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_ENTERPRISE_ID    enterprise;
20    VST_COMMAND_HANDLE   cmd;
21
22    /* get parameters from user */
23    printf("*** Connect Query parameters
         ***\n" );
24    printf("\nEnter Enterprise ID ==>");
25    enterprise = (VST_ENTERPRISE_ID)
         atoi(gets(input));
26
27    /* create the command handle */
28    /* Note that the command handle is
         not */
29    /* destroyed in this routine, but in
         */
30    /* vst_dispatch when final status is
         received. */
31    cmd = VS_Command_Create();
```

```
32    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
33    {
34       /* Send the command to the VolServ
         software. */
35       /* Note that status is not
         processed here. */
36       /* Instead, it is processed in the
         */
37       /* vst_dispatch routine. Also,
         note that */
38       /* default values such as timeout,
         value */
39       /* retry limit and priority are
         set as */
40       /* default parameters. */
41       rc = VSCMD_ConnectQuery(cmd,
42                VSID_QRY_ENTERPRISE_ID,
         enterprise,
43                VSID_ENDFIELD);
44    }
45    return ( rc );
46 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Connect Query request.

The Connect Query command does not trigger unsolicited MediaClass callbacks from VolServ.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, the final status for this command is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive final status on a Connect Query command submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

A client can issue the Connect Query command to determine if an enterprise is already registered. If it is registered, its address is also reported.

- Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.
- Command-specific parameter defaults for the Connect Query command are set with VSCMD_ConnectQuery_SetDefaults. If command-specific defaults are set for the Connect Query command, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Connect Query command, the parameter identifier and the value to be used for the parameter can be submitted on the specific command itself.

The following fields can be retrieved from the status handle after a successful Connect Query command:

- VSID_CONNECT_HANDLE,
- VSID_CONNECT_HANDLE_ENTRY,
- VSID_CONNECT_HANDLE_TABLE,
- VSID_ERROR_CODE,
- VSID_ERROR_CODE_ENTRY,
- VSID_ERROR_CODE_TABLE,
- VSID_QUERY_ENTERPRISE_ID,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_Table(l),
- VS_Command_GetFields(l),
- VS_Connect_GetFields(l),
- VS_Error_GetFields(l),
- VS_Connect_Handle_Table(l),

- VS_Initialize(l),
- VS_Status_GetFields(l),
- VS_Table_GetFields(l),

VSCMD_ Connect-Query_ Set-Defaults

`VSCMD_ConnectQuery_SetDefaults` sets command-level default parameters for Connect Query commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Create Archive Media Class commands are set with `VSCMD_CreateArchiveMediaClass_SetDefaults`. If command-specific defaults are set for Create Archive Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Create Media Class command, the parameter identifier and the value used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_ConnectQuery_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "..." = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for Connect Query commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for Connect Query commands.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for Connect Query commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise whose connection is queried.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Connect Query commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Connect Query commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The total wait time for a command is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Connect Query commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Connect Query commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_ConnectQuery_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_connectquery_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_ConnectQuery API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_connectquery_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_connectquery_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Connect Query default
         parameters ***\n" );
```

```
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_ConnectQuery_SetDefaults(
32          VSID_PRIORITY,
         priority,
33          VSID_USER_FIELD,
         user_field,
34          VSID_TIMEOUT_VALUE,
         timeout,
35          VSID_RETRY_LIMIT,
         retries,
36          VSID_STATUS_WAIT_FLAG,
         wait_flag,
37          VSID_ENTERPRISE_ID,
         enterprise_id,
38          VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_ConnectQuery(l)

VSCMD_ CreateArchiveMediaClass

`VSCMD_CreateArchiveMediaClass` creates an archive media class association. An archive media class association is the association of a MediaClass group with a defined archive.

A MediaClass group establishes common characteristics for the media it contains; no inherent restrictions are placed on where those media can reside within an archive.

Archive media class associations provide the ability to:

- Restrict which archives can contain specific classes/types of media.
- Constrain the number of specific media classes/types that can be associated with any given archive.
- Preclude the erroneous assignment of media into an archive that is incompatible with that media type.
- Ensure an archive has a desired mix of classes of media.
- If needed, preclude media of a known data format from being assigned into an archive that does not have access to a drive compatible with that media type.

Synopsis

```
VST_BOOLEAN VSCMD_CreateArchiveMediaClass
(VST_COMMAND_HANDLE
handle,
"…",
VSID_ENDFIELD)
```

Arguments

- `handle` = The command handle for this Create Archive Media Class command.
- "..." = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_ACTION (VST_ARCHIVE_ACTION_MODE)	Specifies what action VolServ takes when the number of media in the archive media class exceeds the specified high mark threshold. Valid `VSID_ARCHIVE_ACTION` values are enumerated in the *`vs_types.h`* file.
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive to be associated with the archive media class relationship. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CAPACITY (VST_CAPACITY)	Percentage of the total MediaClass capacity that can be stored in this archive.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_COMPONENT_HANDLE_TABLE (VST_TABLE_HANDLE)	Preferred locations (in table format) for media assigned to this archive media class.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.

Parameter Type	Description
VSID_HIGH_MARK (VST_HIGH_MARK)	Percentage of VSID_CAPACITY above which the specified migration policy option is performed or initiated. This field is applicable only if VSID_ARCHIVE_ACTION is set to VSE_ARCHIVE_ACTION_NOTIFY or VSE_ARCHIVE_ACTION_MIG.
VSID_LOW_MARK (VST_LOW_MARK)	Percentage of VSID_CAPACITY below which the specified migration policy option is performed or terminated. This field is applicable only if VSID_ARCHIVE_ACTION is set to VSE_ARCHIVE_ACTION_NOTIFY or VSE_ARCHIVE_ACTION_MIG.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	MediaClass group associated with the archive media class relationship. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MIGRATION_PRIORITY (VST_PRIORITY)	The migration priority to be applied to this archive media class.
VSID_PERCENT (VST_PERCENT)	Percentage of the MediaClass capacity allowed in the archive associated with the archive media class relationship.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The destination archive for media automatically migrated out of this archive media class. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_CreateArchiveMediaClass` returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- `VSE_FALSE` - The command failed. A return code of `VSE_FALSE` (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.
- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code value is `VSE_VOLSERV`, the error occurred in VolServ, and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API, and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.

- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_createarchivemediaclass_execu
       te
4  *
5  * PURPOSE:
6  * This executes the
       VSCMD_CreateArchiveMediaClass
7  * API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_createarchivemediaclass_execu
         te(void)
15 #else
16    VST_BOOLEAN
         vst_createarchivemediaclass_execu
         te()
```

```
17 #endif
18 {
19    int                          i;
20    int                          count;
21    VST_BOOLEAN                  rc =
         VSE_FALSE;
22    VST_ARCHIVE_NAME             archive;
23    VST_MEDIA_CLASS_NAME
         mediaclass;
24    VST_CAPACITY                 capacity;
25    VST_ARCHIVE_ACTION_OPTION    action;
26    VST_HIGH_MARK                highmark;
27    VST_LOW_MARK                 lowmark;
28    VST_PRIORITY                 migpri;
29    VST_ARCHIVE_NAME
         targetarchive;
30    VST_TABLE_HANDLE
         comphandletable;
31    VST_COMPONENT_HANDLE
         comphandle;
32    VST_COMP_TYPE  CompType =
         VSE_COMPTYPE_COLUMN;
33    VST_COMPONENT_ID             CompID;
34    VST_COMMAND_HANDLE           cmd;
35
36    bzero ( CompID, sizeof (
         VST_COMPONENT_ID ) );
37     /* get parameters from user */
38    printf("*** Create Archive Media
         Class parameters ***\n" );
39    printf("Enter Archive Name ==> " );
40    gets( archive );
41    printf("Enter Media Class Name ==> "
         );
42    gets( mediaclass );
43    printf("Enter Capacity Percent ==> "
         );
44    capacity = atoi(gets(input));
45    printf("Enter Archive action option
         (0-none/1-mig/2-notify) ==> " );
46    action = atoi(gets(input));
```

```
47    printf("Enter High Mark Percentage
         ==> " );
48    highmark = atoi(gets(input));
49    printf("Enter Low Mark Percentage ==>
         " );
50    lowmark = atoi(gets(input));
51
52    if ( action == VSE_ARCHIVE_ACTION_MIG
         )
53    {
54       /* these parameters only need to
         be set if */
55       /* the archivemediaclass is being
         set up to */
56       /* support migration. */
57       printf("Enter Target Archive ==> "
         );
58       gets( targetarchive );
59       printf("Enter Migration Priority
         == > " );
60       migpri = atoi(gets(input));
61
         VSCMD_CreateArchiveMediaClass_Set
         Defaults (
62          VSID_TARGET_ARCHIVE_NAME,
         targetarchive,
63          VSID_MIGRATION_PRIORITY,
         migpri,
64          VSID_ENDFIELD );
65    }
66
67    printf("How many prefered placements
         (0 to skip): ");
68    count = atoi(gets(input));
69    if (count > 0)
70    {
71       comphandletable =
         VS_Table_Create(VSE_COMPONENT_HAN
         DLE, count);
72       if (comphandletable ==
         (VST_TABLE_HANDLE) NULL)
73       {
```

```
74          return(VSE_FALSE);
75       }
76       for (i = 0; i < count; i++)
77       {
78          printf("Enter row #%d:", i +
         1);
79          CompID[0] = (short)
         atoi(gets(input));
80          printf("Enter column #%d:", i +
         1);
81          CompID[1] = (short)
         atoi(gets(input));
82          CompID[2] = 0;
83          CompID[3] = 0;
84          comphandle =
         VS_Component_Create();
85
            VS_Component_SetFields(comphandle
            ,
86          VSID_COMP_TYPE, CompType,
87          VSID_COMP_ID, CompID,
88          VSID_ENDFIELD);
89
            VS_Table_AddEntry(comphandletable
            ,comphandle);
90       }
91       /* This also only needs to be set
         if it is */
92       /* actually being used. It is not
         needed */
93       /* otherwise. */
94
            VSCMD_CreateArchiveMediaClass_Set
            Defaults(
95
            VSID_COMPONENT_HANDLE_TABLE,comph
            andletable,
96           VSID_ENDFIELD);
97    }
98
99    /* create the command handle */
```

```
100   /* Note that the command handle is
         not */
101   /* destroyed in this routine, but in
         */
102   /* vst_dispatch when final status is
         received. */
103   cmd = VS_Command_Create();
104   if (cmd != (VST_COMMAND_HANDLE )NULL)
105   {
106      /* Send the command to the VolServ
         software. */
107      /* Note that status is not
         processed here. */
108      /* Instead, it is processed in the
         */
109      /* vst_dispatch routine. Also,
         note that */
110      /* default values such as timeout,
         value */
111      /* retry limit and priority are
         set as */
112      /* default parameters. */
113      rc =
         VSCMD_CreateArchiveMediaClass(cmd
         ,
114               VSID_ARCHIVE_NAME,
         archive,
115               VSID_MEDIA_CLASS_NAME,
         mediaclass,
116               VSID_HIGH_MARK,
         highmark,
117               VSID_LOW_MARK,
         lowmark,
118               VSID_CAPACITY,
         capacity,
119               VSID_ENDFIELD);
120   }
121
122   return ( rc );
123}
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Create Archive Media Class command.

`VSCMD_CreateArchiveMediaClass` does not trigger unsolicited MediaClass callbacks from VolServ.

The migration policy options for are no action, operator notification, and automatic migration.

When the number of media in an archive media class reaches the high mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy option is set to notify.
- Initiates automatic migration of media if the migration policy is set to migrate.

When the number of media in an archive media class drops to the low mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy is set to notify.
- Terminates automatic migration of media if the migration policy is set to migrate.

The capacity value of an archive media class is relative to the MediaClass group specified overall capacity. Consideration should be given to all MediaClass groups that are able to share this archive to ensure that reasonably comparable capacity limitations and high/low marks are set for each archive media class.

Archive media class computed capacity limits are “soft”, that is, they can be exceeded when media are imported or moved in from another archive. If automigration is specified, media of this MediaClass group is then marked for movement to their target archive. The media type capacity designates the “hard” limit when entering media into an archive.

Media can only reside in an archive if their associated MediaClass group has an archive media class assignment in that archive.

An archive media class computed capacity is refigured if the capacity of a MediaClass group changes.

Checks to determine if the high mark has been reached or exceeded or the low mark has been reached or passed occur:

- After any Eject, Enter, Reclassify, or Modify Archive Media Class command executes.
- After the MediaClass group or archive media class association are redefined.

The sum of all archive media class capacities can exceed the archive’s physical ability to house media. If `VSID_CAPACITY` values are set unrealistically high and `VSID_HIGH_MARK` is similarly high, the archive may physically completely fill before any automigration procedure is triggered.

Components listed as preferred for storage of media of this MediaClass group do not have exclusive ownership of those components.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive final status on a Create Archive Media Class request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.
- Command-specific parameter defaults for Create Archive Media Class commands are set with VSCMD_CreateArchiveMediaClass_SetDefaults. If command-specific defaults are set for Create Archive Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Create Media Class command, the parameter identifier and the value used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Create Archive Media Class request:

- VSID_ARCHIVE_NAME,
- VSID_COMPONENT_HANDLE,
- VSID_COMPONENT_HANDLE_ENTRY,

- VSID_COMPONENT_HANDLE_TABLE,
- VSID_ERROR_CODE,
- VSID_ERROR_CODE_ENTRY,
- VSID_ERROR_CODE_TABLE,
- VSID_MEDIA_CLASS_NAME,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_CreateArchiveMediaClass_SetDefaults(l),
- VSCMD_DeleteArchiveMediaClass(l),
- VSCMD_ModifyArchiveMediaClass(l)

VSCMD_CreateArchiveMediaClass_SetDefaults

`VSCMD_CreateArchvieMediaClass_SetDefaults` sets the command-level default parameters for Create Archive Media Class commands.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Create Media Class commands are set with `VSCMD_CreateMediaClass_SetDefaults`. If command-specific defaults are set for Create Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Create Media Class command, the parameter identifier and the value used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_CreateArchive
MediaClass_SetDefaults
(
"…",
VSID_ENDFIELD)
```

Arguments

- "..." = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_ACTION (VST_ARCHIVE_ACTION_MODE)	Specifies what action VolServ takes when the number of media in the archive media class exceeds the specified high mark threshold. Valid VSID_ARCHIVE_ACTION values are enumerated in the *vs_types.h* file.
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive to be associated with the archive media class relationship. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CAPACITY (VST_CAPACITY)	Percentage of the total MediaClass capacity that can be stored in this archive.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for Create Archive Media Class commands.
VSID_COMPONENT_HANDLE_TABLE (VST_TABLE_HANDLE)	Preferred locations (in table format) for media assigned to this archive media class.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for Create Archive Media Class commands.

Parameter Type	Description
VSID_HIGH_MARK (VST_HIGH_MARK)	Percentage of VSID_CAPACITY above which the specified migration policy option is performed or initiated. This field is applicable only if VSID_ARCHIVE_ACTION is set to VSE_ARCHIVE_ACTION_NOTIFY or VSE_ARCHIVE_ACTION_MIG.
VSID_LOW_MARK (VST_LOW_MARK)	Percentage of VSID_CAPACITY below which the specified migration policy option is performed or terminated. This field is applicable only if VSID_ARCHIVE_ACTION is set to VSE_ARCHIVE_ACTION_NOTIFY or VSE_ARCHIVE_ACTION_MIG.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	MediaClass group associated with the archive media class relationship. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MIGRATION_PRIORITY (VST_PRIORITY)	The migration priority to be applied to this archive media class.
VSID_PERCENT (VST_PERCENT)	Percentage of the MediaClass group capacity allowed in the archive associated with the archive media class relationship.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Create Archive Media Class commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Create Archive Media Class commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The destination archive for media automatically migrated out of this archive media class. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The total wait time for a command is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Create Archive Media Class commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Create Archive Media Class commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_CreateArchiveMediaClass_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_createarchivemediaclass_execu
       te
4  *
5  * PURPOSE:
6  * This executes the
       VSCMD_CreateArchiveMediaClass
7  * API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
       vst_createarchivemediaclass_execu
       te(void)
15 #else
```

```
16    VST_BOOLEAN
         vst_createarchivemediaclass_execu
         te()
17 #endif
18 {
19    int                        i;
20    int                        count;
21    VST_BOOLEAN                rc =
         VSE_FALSE;
22    VST_ARCHIVE_NAME           archive;
23    VST_MEDIA_CLASS_NAME
         mediaclass;
24    VST_CAPACITY               capacity;
25    VST_ARCHIVE_ACTION_OPTION  action;
26    VST_HIGH_MARK              highmark;
27    VST_LOW_MARK               lowmark;
28    VST_PRIORITY               migpri;
29    VST_ARCHIVE_NAME
         targetarchive;
30    VST_TABLE_HANDLE
         comphandletable;
31    VST_COMPONENT_HANDLE
         comphandle;
32    VST_COMP_TYPE  CompType =
         VSE_COMPTYPE_COLUMN;
33    VST_COMPONENT_ID           CompID;
34    VST_COMMAND_HANDLE         cmd;
35
36    bzero ( CompID, sizeof (
         VST_COMPONENT_ID ) );
37     /* get parameters from user */
38    printf("*** Create Archive Media
         Class parameters ***\n" );
39    printf("Enter Archive Name ==> " );
40    gets( archive );
41    printf("Enter Media Class Name ==> "
         );
42    gets( mediaclass );
43    printf("Enter Capacity Percent ==> "
         );
44    capacity = atoi(gets(input));
```

```
45    printf("Enter Archive action option
         (0-none/1-mig/2-notify) ==> " );
46    action = atoi(gets(input));
47    printf("Enter High Mark Percentage
         ==> " );
48    highmark = atoi(gets(input));
49    printf("Enter Low Mark Percentage ==>
         " );
50    lowmark = atoi(gets(input));
51
52    if ( action == VSE_ARCHIVE_ACTION_MIG
         )
53    {
54       /* these parameters only need to
         be set if */
55       /* the archivemediaclass is being
         set up to */
56       /* support migration. */
57       printf("Enter Target Archive ==> "
         );
58       gets( targetarchive );
59       printf("Enter Migration Priority
         == > " );
60       migpri = atoi(gets(input));
61
         VSCMD_CreateArchiveMediaClass_Set
         Defaults (
62          VSID_TARGET_ARCHIVE_NAME,
         targetarchive,
63          VSID_MIGRATION_PRIORITY,
         migpri,
64          VSID_ENDFIELD );
65    }
66
67    printf("How many prefered placements
         (0 to skip): ");
68    count = atoi(gets(input));
69    if (count > 0)
70    {
71       comphandletable =
         VS_Table_Create(VSE_COMPONENT_HAN
         DLE, count);
```

```
72       if (comphandletable ==
         (VST_TABLE_HANDLE) NULL)
73       {
74          return(VSE_FALSE);
75       }
76       for (i = 0; i < count; i++)
77       {
78          printf("Enter row #%d:", i +
         1);
79          CompID[0] = (short)
         atoi(gets(input));
80          printf("Enter column #%d:", i +
         1);
81          CompID[1] = (short)
         atoi(gets(input));
82          CompID[2] = 0;
83          CompID[3] = 0;
84          comphandle =
         VS_Component_Create();
85
         VS_Component_SetFields(comphandle
         ,
86          VSID_COMP_TYPE, CompType,
87          VSID_COMP_ID, CompID,
88          VSID_ENDFIELD);
89
         VS_Table_AddEntry(comphandletable
         ,comphandle);
90       }
91       /* This also only needs to be set
         if it is */
92       /* actually being used. It is not
         needed */
93       /* otherwise. */
94
         VSCMD_CreateArchiveMediaClass_Set
         Defaults(
95
         VSID_COMPONENT_HANDLE_TABLE,
         comphandletable,
96             VSID_ENDFIELD);
97    }
```

```
98
99    /* create the command handle */
100   /* Note that the command handle is
         not */
101   /* destroyed in this routine, but in
         */
102   /* vst_dispatch when final status is
         received. */
103   cmd = VS_Command_Create();
104   if (cmd != (VST_COMMAND_HANDLE )NULL)
105   {
106      /* Send the command to the VolServ
         software. */
107      /* Note that status is not
         processed here. */
108      /* Instead, it is processed in the
         */
109      /* vst_dispatch routine. Also,
         note that */
110      /* default values such as timeout,
         value */
111      /* retry limit and priority are
         set as */
112      /* default parameters. */
113      rc =
         VSCMD_CreateArchiveMediaClass(cmd
         ,
114               VSID_ARCHIVE_NAME,
         archive,
115               VSID_MEDIA_CLASS_NAME,
         mediaclass,
116               VSID_HIGH_MARK,
         highmark,
117               VSID_LOW_MARK,
         lowmark,
118               VSID_CAPACITY,
         capacity,
119               VSID_ENDFIELD);
120   }
121
122   return ( rc );
123}
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_CreateArchiveMediaClass(l)

VSCMD_ CreateMedia-Class

`VSCMD_CreateMediaClass` creates a new MediaClass group.

A MediaClass group establishes common characteristics for the media it contains.

Synopsis

```
VST_BOOLEAN VSCMD_CreateMediaClass
(
VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)
```

Arguments

- `handle` = The command handle for the Create Media Class request.
- `"…"` = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = `Re`quired at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CAPACITY (VST_CAPACITY)	Maximum number of media allowed in this MediaClass group.
VSID_CLASS_MOUNT_STATE (VST_CLASS_MOUNT_STATE)	Indicates whether this MediaClass group supports the “mount by MediaClass” functionality. Valid `VSID_CLASS_MOUNT_STATE` values are enumerated in the *vs_types.h* file.
VSID_CLASS_RPC_OPTION (VST_CLASS_RPC_OPTION)	Indicates whether callbacks are activated for this MediaClass group, and if they are, which callback scheme is used. Valid `VSID_CLASS_RPC_OPTION` values are enumerated in the *vs_types.h* file.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.
VSID_HIGH_MARK (VST_HIGH_MARK)	Percentage of `VSID_CAPACITY` above which the specified migration policy option is performed or initiated. This field is applicable only if `VSID_ARCHIVE_ACTION` is set to `VSE_ARCHIVE_ACTION_NOTIFY` or `VSE_ARCHIVE_ACTION_MIG`.
VSID_HOST_NAME (VST_HOST_NAME)	Network-assigned name of the computer where the task that “listens” for MediaClass callbacks executes. Applicable only if `VSID_CLASS_RPB_OPTION` is set to `VSE_CLASS_RPC_STANDARD`.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Unique name assigned to the new MediaClass group.

Parameter Type	Description
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	Media type supported by this MediaClass group. Valid media type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_NOTIFY_COMMENT (VST_NOTIFY_COMMENT)	User-specified comment included in a system log message when the number of media in the MediaClass group exceeds the high mark threshold. MediaClass name, fill level, high mark threshold, and capacity values are automatically included in the system log message and need not be included in VSID_NOTIFY_COMMENT.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	RPC procedure number of the client process to receive callbacks generated for this MediaClass group. VSID_PROCEDURE_NUMBER is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_PROCEDURE_NUMBER is not applicable.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	RPC program number of the client process to receive callbacks generated for this MediaClass group. VSID_PROGRAM_NUMBER is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_PROGRAM_NUMBER is not applicable.

Parameter Type	Description
VSID_PROTOCOL (VST_PROTOCOL)	Internet protocol VolServ uses to send callbacks for this MediaClass group. The default VSID_PROTOCOL is VSE_PROT_TCP. VSID_PROTOCOL is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_PROTOCOL is not applicable.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise to receive callbacks for this MediaClass group.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	RPC version number of the client process to receive callbacks generated for this MediaClass group. VSID_VERSION_NUMBER is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_VERSION_NUMBER is not applicable.

Return Values

VSCMD_CreateMediaClass returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
    - If the object code value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
    - If the object code value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_createmediaclass_execute
4  *
5  * PURPOSE:
6  * This executes the
       VSCMD_CreateMediaClass API call.
7  *
8  * PARAMETERS:
9  * none
10
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_createmediaclass_execute(void
         )
14 #else
15    VST_BOOLEAN
         vst_createmediaclass_execute()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIA_CLASS_NAME    mediaclass;
20    VST_MEDIA_TYPE_NAME
         MediaTypeName;
21    VST_CAPACITY            capacity;
22    VST_CLASS_MOUNT_STATE   mountstate;
23    VST_HIGH_MARK           highmark;
24    VST_COMMAND_HANDLE      cmd;
25    VST_NOTIFY_COMMENT      comment;
26    VST_CLASS_RPC_OPTION    rpc_option;
27    VST_HOSTNAME
         rpc_hostname;
28    VST_PROGRAM_NUMBER      rpc_prognum;
29    VST_VERSION_NUMBER      rpc_versnum;
30    VST_PROCEDURE_NUMBER    rpc_procnum;
31    VST_PROTOCOL
         rpc_protocol;
```

```
32    VST_ENTERPRISE_ID
         enterpriseid;
33
34    /* get parameters from user */
35    printf("*** Create Media Class
         parameters ***\n" );
36    printf("\nEnter Media Class Name
         ==>");
37    gets( mediaclass);
38    printf("\nEnter Media Type Name ==>
         ");
39    gets( MediaTypeName);
40    printf("\nEnter capacity==>");
41    capacity = atoi(gets(input));
42    printf("\nEnter class mount state (0)
         no, (1) ok: ");
43    mountstate = atoi(gets(input));
44    printf("\nEnter high mark ==>");
45    highmark = atoi(gets(input));
46    printf("\nEnter notify comment
         ==>");
47    gets(comment);
48    printf("\nEnter Option (0) no
         callbacks, (1) standard, (2)
         Enterprise==>");
49    rpc_option = (VST_CLASS_RPC_OPTION)
         atoi(gets(input));
50    if (rpc_option ==
         VSE_CLASS_RPC_NONE)
51    {
52
         VSCMD_CreateMediaClass_SetDefault
         s(
53                VSID_CLASS_RPC_OPTION,
         rpc_option,
54                VSID_ENDFIELD);
55    }
56    else if (rpc_option ==
         VSE_CLASS_RPC_STANDARD)
57    {
58       printf("\nEnter RPC Host Name
         ==>");
```

```
59        gets(rpc_hostname);
60        printf("\nEnter program number
          ==>");
61        rpc_prognum =
          (VST_PROGRAM_NUMBER)
          atol(gets(input));
62        printf("\nEnter version number
          ==>");
63        rpc_versnum =
          (VST_PROGRAM_NUMBER)
          atol(gets(input));
64        printf("\nEnter procedure number
          ==>");
65        rpc_procnum =
          (VST_PROGRAM_NUMBER)
          atol(gets(input));
66        printf("\nEnter Protocol (6) TCP
          or (17) UDP ==>");
67        rpc_protocol = (VST_PROTOCOL)
          atoi(gets(input));
68
          VSCMD_CreateMediaClass_SetDefault
          s(
69              VSID_HOST_NAME,
          rpc_hostname,
70              VSID_CLASS_RPC_OPTION,
          rpc_option,
71              VSID_PROGRAM_NUMBER,
          rpc_prognum,
72              VSID_VERSION_NUMBER,
          rpc_versnum,
73              VSID_PROCEDURE_NUMBER,
          rpc_procnum,
74              VSID_PROTOCOL,
          rpc_protocol,
75              VSID_ENDFIELD);
76    }
77    else if (rpc_option ==
          VSE_CLASS_RPC_ENTERPRISE)
78    {
79        printf("\nEnter enterprise id
          ==>");
```

```
80          enterpriseid =
            (VST_ENTERPRISE_ID)
            atol(gets(input));
81
            VSCMD_CreateMediaClass_SetDefault
            s(
82             VSID_CLASS_RPC_OPTION,
            rpc_option,
83             VSID_TARGET_ENTERPRISE_ID,
            enterpriseid,
84             VSID_ENDFIELD);
85       }
86
87       /* create the command handle */
88       /* Note that the command handle is
            not */
89       /* destroyed in this routine, but in
            */
90       /* vst_dispatch when final status is
            received. */
91       cmd = VS_Command_Create();
92       if ( cmd != (VST_COMMAND_HANDLE)
            NULL)
93       {
94          /* Send the command to the VolServ
            software. */
95          /* Note that status is not
            processed here. */
96          /* Instead, it is processed in the
            */
97          /* vst_dispatch routine. Also,
            note that */
98          /* default values such as timeout,
            value */
99          /* retry limit and priority are
            set as */
100         /* default parameters. */
101         rc = VSCMD_CreateMediaClass(cmd,
102            VSID_MEDIA_CLASS_NAME,
            mediaclass,
103            VSID_MEDIA_TYPE_NAME,
            MediaTypeName,
```

```
104         VSID_CAPACITY,
          capacity,
105         VSID_CLASS_MOUNT_STATE,
          mountstate,
106         VSID_HIGH_MARK,
          highmark,
107         VSID_NOTIFY_COMMENT,
          comment,
108         VSID_ENDFIELD);
109    }
110    return ( rc );
111}
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Create Media Class request.

The Create Media Class command does not trigger unsolicited MediaClass callbacks from VolServ.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

Name specified for the MediaClass group must be unique. Any requests to create non-unique MediaClass names fail.

MediaClass groups can span archives.

MediaClass groups may contain only one type of media.

Checks to determine if `VSID_HIGH_MARK` has been reached or exceeded occur after any Reclassify, Import, or Modify Media Class command.

VSID_CAPACITY is a “hard” limit. VolServ does not permit the number of media assigned to a MediaClass group to exceed the VSID_CAPACITY for that MediaClass group.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive final status on a Create Media Class request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.
- Command-specific parameter defaults for Create Media Class commands are set with VSCMD_CreateMediaClass_SetDefaults. If command-specific defaults are set for Create Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Create Media Class command, the parameter identifier and the value used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Create Media Class command:

- VSID_MEDIA_CLASS_NAME,

- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_CreateMediaClass_SetDefaults(l),
- VSCMD_DeleteMediaClass(l),
- VSCMD_ModifyMediaClass(l)

VSCMD_ CreateMedia-Class_ SetDefaults

`VSCMD_CreateMediaClass_SetDefaults` sets the command-level default parameters for Create Media Class commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Create Media Class commands are set with `VSCMD_CreateMediaClass_SetDefaults`. If command-specific defaults are set for Create Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Create Media Class command, the parameter identifier and the value used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_CreateMediaClass
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "..." = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CAPACITY (VST_CAPACITY)	Maximum number of media allowed in this MediaClass group.
VSID_CLASS_MOUNT_STATE (VST_CLASS_MOUNT_STATE)	Indicates whether this MediaClass group supports the "mount by MediaClass" functionality. Valid VSID_CLASS_MOUNT_STATE values are enumerated in the *vs_types.h* file.
VSID_CLASS_RPC_OPTION (VST_CLASS_RPC_OPTION)	Indicates whether callbacks are to be activated for this MediaClass group, and if they are, which callback scheme is used. Valid VSID_CLASS_RPC_OPTION values are enumerated in the *vs_types.h* file.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for Create Media Class commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on Create Media Class commands.

Parameter Type	Description
VSID_HIGH_MARK (VST_HIGH_MARK)	Percentage of VSID_CAPACITY above which the specified migration policy option is performed or initiated. This field is applicable only if VSID_ARCHIVE_ACTION is set to VSE_ARCHIVE_ACTION_NOTIFY or VSE_ARCHIVE_ACTION_MIG.
VSID_HOST_NAME (VST_HOST_NAME)	Network-assigned name of the computer where the task that “listens” for MediaClass callbacks executes. Applicable only if VSID_CLASS_RPB_OPTION is set to VSE_CLASS_RPC_STANDARD.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Unique name to be assigned to the new MediaClass group.
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	Media type supported by this MediaClass group. Valid media type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_NOTIFY_COMMENT (VST_NOTIFY_COMMENT)	User-specified comment to be included in a system log message when the number of media in the MediaClass group exceeds the high mark threshold. MediaClass name, fill level, high mark threshold, and capacity values are automatically included in the system log message and need not be included in VSID_NOTIFY_COMMENT.
VSID_PRIORITY (VST_PRIORITY)	Requested execution priority for Create Media Class commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	RPC procedure number of the client process to receive callbacks generated for this MediaClass group. VSID_PROCEDURE_NUMBER is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_PROCEDURE_NUMBER is not applicable.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	RPC program number of the client process to receive callbacks generated for this MediaClass group. VSID_PROGRAM_NUMBER is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_PROGRAM_NUMBER is not applicable.
VSID_PROTOCOL (VST_PROTOCOL)	Internet protocol VolServ uses to send callbacks for this MediaClass group. The default VSID_PROTOCOL is VSE_PROT_TCP. VSID_PROTOCOL is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_PROTOCOL is not applicable.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Create Media Class commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Create Media Class commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise to receive callbacks for this MediaClass group.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a timeout to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Create Media Class commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Create Media Class commands. Neither the API software nor VolServ uses USER_FIELD.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	RPC version number of the client process to receive callbacks generated for this MediaClass group. VSID_VERSION_NUMBER is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_VERSION_NUMBER is not applicable.

Return Values

`VSCMD_CreateMediaClass_SetDefaults` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_createmediaclass_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_CreateMediaClass API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_createmediaclass_defaults(voi
         d)
15 #else
16    VST_BOOLEAN
         vst_createmediaclass_defaults()
```

```
#endif
{
   VST_BOOLEAN             rc =
      VSE_FALSE;
   VST_PRIORITY            priority;
   VST_USER_FIELD          user_field;
   VST_TIME_OUT            timeout;
   VST_RETRY_LIMIT         retries;
   VST_STATUS_WAIT_FLAG    wait_flag;
   VST_ENTERPRISE_ID
      enterprise_id;

   /* get parameters from user */
   printf("*** Create Archive Media
      Class default parameters ***\n" );
   vst_promptforglobals(&priority,
      user_field, &timeout, &retries,
      &wait_flag, &enterprise_id);
   /* set the default parameters */
   rc =
      VSCMD_CreateMediaClass_SetDefault
      s(
            VSID_PRIORITY,
      priority,
            VSID_USER_FIELD,
      user_field,
            VSID_TIMEOUT_VALUE,
      timeout,
            VSID_RETRY_LIMIT,
      retries,
            VSID_STATUS_WAIT_FLAG,
      wait_flag,
            VSID_ENTERPRISE_ID,
      enterprise_id,
            VSID_ENDFIELD);
   return ( rc );
}
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_CreateMediaClass(l),
- VSCMD_DeleteMediaClass(l),
- VSCMD_ModifyMediaClass(l)

VSCMD_Drive Vary

`VSCMD_DriveVary` is issued to execute VolServ Drive Vary requests and to change the operational availability of a drive.

The drive and target state (`VSE_COMP_ONLINE`, `VSE_COMP_OFFLINE`, or `VSE_COMP_DIAGNOSTIC`) must be specified.

A drive in the off-line, unavailable, or diagnostic state is excluded from VolServ's drive selection algorithm.

A Mount or Lock request for an off-line, unavailable, or diagnostic drive fails.

Conversely, varying a drive to the on-line state makes it available for selection for Mount or Lock requests.

Upon receipt of a Drive Vary request, VolServ attempts to change the state of the specified drive. The return code presented to the client indicates the success or failure of the command.

Synopsis

```
VST_BOOLEAN VSCMD_DriveVary
( VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD )
```

Arguments

- `handle` = The command handle for this Drive Vary request.
- `"…"` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_COMP_STATE (VST_COMP_STATE)	The target state for the individual drive specified in VSID_DRIVE_ID or drive pool group given in VSID_DRIVEPOOL_NAME. Used when varying all drives to the same state. Valid VSID_COMP_STATE values are enumerated in the *vs.types.h* file.
VSID_COMP_STATE_LIST (int)	Number of states contained in this list.
(VST_COMP_STATE *)	Pointer to the list of one or more target states for the drives specified in VSID_DRIVE_ID_LIST. Used when varying the drives to different states. Valid VSID_COMP_STATE_LIST values are enumerated in the vs_types.h file.
VSID_DRIVE_ID (VST_DRIVE_ID)	An individual drive whose state is varied. Not necessary when specifying a drive list or drive pool.
VSID_DRIVE_ID_LIST (int)	Number of drives contained in this list used with VSID_COMP_STATE_LIST.
(VST_DRIVE_ID *)	Pointer to a list of one or more drives whose states are to be varied. Not necessary when specifying a drive list or drive identifier.

Parameter Type	Description
VSID_DRIVEPOOL_NAME (VST_DRIVEPOOL_NAME)	Name of a drive pool group to vary the state of all drives associated with the drive pool group. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted. Not necessary when specifying a drive list or drive identifier.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this command. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software is to retry for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE`.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The default time-out value is 120 seconds.

Return Values

VSCMD_DriveVary returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.
  - To determine where the error occurred and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.

- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_drivevary_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_DriveVary API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_drivevary_execute(void)
14 #else
15    VST_BOOLEAN vst_drivevary_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    int                  count;
20    VST_DRIVE_ID
         drivelist[VST_MAX_ITEMS];
21    VST_DRIVE_ID         temp_drive_id;
22    VST_COMP_STATE       temp_state;
```

```
23    VST_COMP_STATE
         statelist[VST_MAX_ITEMS];
24    VST_DRIVE_POOL_NAME  poolname;
25    int                  i;
26    VST_COMMAND_HANDLE   cmd;
27    int                  varyopt;
28    int                  stateopt;
29
30    /* get parameters from user */
31    printf("*** Drive Vary parameters
         ***\n" );
32    printf("0) Vary by drive list , 1)
         Vary by drive pool 2) Vary by
         drive ID ==> " );
33    varyopt = atoi(gets(input));
34
35    if (varyopt == 0)
36    {
37       /* vary by drive list */
38       /* get the list */
39       count =
         vst_getdrivelist(drivelist,
         VST_MAX_ITEMS);
40       VSCMD_DriveVary_SetDefaults(
41                VSID_DRIVE_ID_LIST,
         count,drivelist,
42                VSID_ENDFIELD);
43    }
44    else if (varyopt == 1)
45    {
46       /* vary by drive pool */
47
         return(vst_drivevary_pool_execute
         ());
48    }
49    else
50    {
51       /* vary a single drive */
52       printf("\nEnter Drive ID ==> ");
53       temp_drive_id =
         atoi(gets(input));
54       VSCMD_DriveVary_SetDefaults(
```

```
55                   VSID_DRIVE_ID,
      temp_drive_id,
56                   VSID_ENDFIELD);
57    }
58
59    printf("0) Vary by state list , 1)
         Vary single state ==> " );
60    stateopt = atoi(gets(input));
61    if (stateopt == 0)
62    {
63       /* vary by using a list of
         component states */
64       printf("\nEnter New States
         (1)ONLINE (2)OFFLINE (3) DIAG");
65       for (i = 0; i < count; i++)
66       {
67          printf("State #%d: ",
         count+1);
68          statelist[i] =
         atoi(gets(input));
69       }
70
71       VSCMD_DriveVary_SetDefaults(
72             VSID_COMP_STATE_LIST,
         count,statelist,
73             VSID_ENDFIELD);
74    }
75    else
76    {
77       /* vary everyting to a single
         state */
78       printf("\nEnter New State (1)
         ONLINE (2) OFFLINE (3) DIAG ==>");
79       temp_state = atoi(gets(input));
80       VSCMD_DriveVary_SetDefaults(
81                   VSID_COMP_STATE,
         temp_state,
82                   VSID_ENDFIELD);
83    }
84
85    /* create the command handle */
```

```
86    /* Note that the command handle is
         not */
87    /* destoyed in this routine, but in
         */
88    /* vst_dispatch when final status is
         received. */
89    cmd = VS_Command_Create();
90    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
91    {
92       /* Send the command to the VolServ
         software. */
93       /* Note that status is not
         processed here. */
94       /* Instead, it is processed in the
         */
95       /* vst_dispatch routine. Also,
         note that */
96       /* default values such as timeout,
         value */
97       /* retry limit and priority are
         set as */
98       /* default parameters. */
99       rc = VSCMD_DriveVary(cmd,
100                          VSID_ENDFIELD);
101   }
102   return ( rc );
103}
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generate intermediate status in response to a Drive Vary request.

`VSCMD_DriveVary` does not trigger any MediaClass callbacks from VolServ.

If a list of media specified in a Drive Vary request contains media of more than one type, the request fails.

Mounted drives that have their state changed remain in-use; varying a drive has no impact on client data transfer operations in progress, and the client receives no automatic notification of a drive state change.

Drives can be varied regardless of whether or not they are associated with an archive.

Drives can be varied regardless of whether or not they are allocated; however, allocated drives that are not `on-line` cannot be dismounted.

Drives can be varied by an operator and over the client interface into the `off-line`, `on-line`, and `diagnostic` states only.

The `unavailable` state is only assignable by VolServ when a higher level component in the archive system is no longer `on-line`. For example, varying a CLM `off-line` causes the associated drive to be viewed as `unavailable`.

The `VSID_DRIVE_ID_LIST` and `VSID_COMP_STATE_LIST` parameters require that two arguments be passed instead of one.

All parameters can be set for the specific request being sent by passing them to this function, or they can be set for all Drive Vary requests using the `VSCMD_DriveVary_SetDefaults` command.

It is possible to vary drives to different states with one request. To do this, use the `VSID_DRIVE_ID_LIST` and `VSID_COMP_STATE_LIST` parameters.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive intermediate and final status on command requests submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.
- Command-specific parameter defaults for the Drive Vary command are set with VSCMD_DriveVary_SetDefaults. If command-specific defaults are set for the Drive Vary command, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Drive Vary command, the parameter identifier and the value to be used for the parameter can be submitted on the command request itself.

The following fields can be retrieved from the status handle after a successful DriveVary request:

- VSID_ERROR_CODE,
- VSID_ERROR_CODE_ENTRY,
- VSID_ERROR_CODE_TABLE,
- VSID_DRIVE_ID,

- VSID_DRIVE_ID_ENTRY,
- VSID_DRIVE_ID_TABLE,
- VSID_MEDIA_ID,
- VSID_MEDIA_ID_ENTRY,
- VSID_MEDIA_ID_TABLE,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE, VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_DriveVary_SetDefaults(l)

VSCMD_DriveVary_SetDefaults

`VSCMD_DriveVary_SetDefaults` is the call issued to set the command default parameters for Drive Vary commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

Synopsis

```
VST_BOOLEAN VSCMD_DriveVary_SetDefaults
(
"…",
VSID_ENDFIELD )
```

Arguments

- `"…"` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` =Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_COMP_STATE (VST_COMP_STATE)	The target state for the individual drive or drive pool group specified in `VSID_DRIVE_ID`. Used when varying the drives to different states. Valid `VSID_COMP_STATE` values are enumerated in the *vs_types.h* file.
VSID_COMP_STATE_LIST (int)	Number of states contained in this list.

Parameter Type	Description
(VST_COMP_STATE *)	Pointer to the list of one or more target states for the drive specified in VSID_DRIVE_ID_LIST. Used when varying the drives to different states. Valid VSID_COMP_STATE_LIST values are enumerated in the *vs_types.h* file.
VSID_DRIVE_ID (VST_DRIVE_ID)	An individual drive whose state is varied. Not necessary when specifying drives to different states.
VSID_DRIVE_ID_LIST (int)	Number of drives contained in this list.
(VST_DRIVE_ID *)	Pointer to the list of one or more drives whose states are to be varied. Not necessary when specifying drives to different states.
VSID_DRIVEPOOL_NAME (VST_DRIVEPOOL_NAME)	Name of a drive pool group to vary the state of all drives associated with the drive pool group. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted. Not necessary when specifying drives to different states.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for Drive Vary commands.
VSID_PRIORITY (VST_PRIORITY)	The execution priority (to override the default global execution priority) for Drive Vary command requests. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software is to retry for command status from VolServ before returning a time-out to the client software (to override the default global retry limit) for Drive Vary command requests. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in the USER_FIELD for Drive Vary commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Drive Vary commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_DriveVary_SetDefaults` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_drivevary_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_DriveVary API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_drivevary_execute(void)
14 #else
15    VST_BOOLEAN vst_drivevary_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    int                  count;
20    VST_DRIVE_ID
         drivelist[VST_MAX_ITEMS];
```

```
21    VST_DRIVE_ID         temp_drive_id;
22    VST_COMP_STATE       temp_state;
23    VST_COMP_STATE
         statelist[VST_MAX_ITEMS];
24    VST_DRIVE_POOL_NAME  poolname;
25    int                  i;
26    VST_COMMAND_HANDLE   cmd;
27    int                  varyopt;
28    int                  stateopt;
29
30    /* get parameters from user */
31    printf("*** Drive Vary parameters
         ***\n" );
32    printf("0) Vary by drive list , 1)
         Vary by drive pool 2) Vary by
         drive ID ==> " );
33    varyopt = atoi(gets(input));
34
35    if (varyopt == 0)
36    {
37       /* vary by drive list */
38       /* get the list */
39       count =
         vst_getdrivelist(drivelist,
         VST_MAX_ITEMS);
40       VSCMD_DriveVary_SetDefaults(
41                VSID_DRIVE_ID_LIST,
         count,drivelist,
42                VSID_ENDFIELD);
43    }
44    else if (varyopt == 1)
45    {
46       /* vary by drive pool */
47
         return(vst_drivevary_pool_execute
         ());
48    }
49    else
50    {
51       /* vary a single drive */
52       printf("\nEnter Drive ID ==> ");
```

```
53       temp_drive_id =
         atoi(gets(input));
54       VSCMD_DriveVary_SetDefaults(
55                   VSID_DRIVE_ID,
         temp_drive_id,
56                   VSID_ENDFIELD);
57    }
58
59    printf("0) Vary by state list , 1)
         Vary single state ==> " );
60    stateopt = atoi(gets(input));
61    if (stateopt == 0)
62    {
63       /* vary by using a list of
         component states */
64       printf("\nEnter New States
         (1)ONLINE (2)OFFLINE (3) DIAG");
65       for (i = 0; i < count; i++)
66       {
67          printf("State #%d: ",
         count+1);
68          statelist[i] =
         atoi(gets(input));
69       }
70
71       VSCMD_DriveVary_SetDefaults(
72             VSID_COMP_STATE_LIST,
         count,statelist,
73             VSID_ENDFIELD);
74    }
75    else
76    {
77       /* vary everyting to a single
         state */
78       printf("\nEnter New State (1)
         ONLINE (2) OFFLINE (3) DIAG ==>");
79       temp_state = atoi(gets(input));
80       VSCMD_DriveVary_SetDefaults(
81                   VSID_COMP_STATE,
         temp_state,
82                   VSID_ENDFIELD);
83    }
```

```
84
85    /* create the command handle */
86    /* Note that the command handle is
         not */
87    /* destroyed in this routine, but in
         */
88    /* vst_dispatch when final status is
         received. */
89    cmd = VS_Command_Create();
90    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
91    {
92       /* Send the command to the VolServ
         software. */
93       /* Note that status is not
         processed here. */
94       /* Instead, it is processed in the
         */
95       /* vst_dispatch routine. Also,
         note that,*/
96       /* default values such as timeout,
         value */
97       /* retry limit and priority are
         set as */
98       /* default parameters. */
99       rc = VSCMD_DriveVary(cmd,
100                            VSID_ENDFIELD);
101   }
102   return ( rc );
103}
```

Notes

The `VSID_DRIVE_ID_LIST` and `VSID_COMP_STATE_LIST` parameters require that two arguments be passed instead of one.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_DriveVary(l)

VSCMD_ Export

`VSCMD_Export` is issued to execute the VolServ Export request.

A client uses the Export request to mark media and related information for removal from the VolServ system. If the specified media are not associated with an archive, they are logically removed from the VolServ system. If the specified media are associated with an archive, they are placed on the eject list of the appropriate archive.

A client can also use the Export request to remove information about media that have been checked out of the archive and are physically out of the archive.

Upon receipt of an Export request, VolServ marks the specified media for ejection and returns a successful return code to the client. A message is sent to the operator console indicating which media need to be ejected from the archive.

To physically remove media from the archive system, an operator must select the eject function from the appropriate archive's console display. The eject function is not available from the API.

After a medium specified on an Export request is physically removed from the archive system, the medium is no longer under the control of VolServ. Consequently, all information related to exported medium is deleted from the VolServ system.

Synopsis

```
VST_BOOLEAN VSCMD_Export (
VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD )
```

Arguments

- `handle` = The command handle for this Export request.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_COMMENT (VST_COMMENT)	The export comment to use for these media. This comment appears with the media on the archive's eject list.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.
VSID_MEDIA_ID_LIST (int)	Number of media identified in the list.
(char **)	Identifiers of the media to export.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software is to retry for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_Export` returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- `VSE_FALSE` - The command failed. A return code of `VSE_FALSE` (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code's value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.
- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code's value is `VSE_VOLSERV`, the error occurred in VolServ, and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code's value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API, and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.

- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_export_execute
4  *
5  * PURPOSE:
6  * This function sends an export command
       to the
7  * Volserv, prompting for all values
       needed.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN vst_export_execute(void)
15 #else
16    VST_BOOLEAN vst_export_execute()
17 #endif
18 {
19    VST_BOOLEAN          rc = VSE_FALSE;
20    int                  count;
21    char                 *
         medialist[VST_MAX_ITEMS];
22    VST_COMMENT          comment;
```

```
23    VST_COMMAND_HANDLE   cmd;
24
25    /* get parameters from user */
26    printf("*** Export Parameters ***\n"
         );
27    printf("\nEnter Export Comment ");
28    gets( comment);
29    count =
         vst_getmedialist(medialist,
         VST_MAX_ITEMS);
30    /* create the command handle */
31    /* Note that the command handle is
         not */
32    /* destroyed in this routine, but in
         */
33    /* vst_dispatch when final status is
         received. */
34    cmd = VS_Command_Create();
35    /* validate the command handle */
36    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
37    {
38       /* Send the command to the VolServ
         software. */
39       /* Note that status is not
         processed here. */
40       /* Instead, it is processed in the
         */
41       /* vst_dispatch routine. Also,
         note that */
42       /* default values such as time
         out, value */
43       /* retry limit and priority are
         set as */
44       /* default parameters. */
45       rc = VSCMD_Export(cmd,
46                VSID_COMMENT, comment,
47                VSID_MEDIA_ID_LIST,
         count, medialist,
48                VSID_ENDFIELD);
49    }
50
```

```
51    return ( rc );
52 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ does not generate intermediate status in response to an Export request.

`VSCMD_Export` triggers MediaClass callbacks from VolServ.

The `VSID_Media_ID_LIST` parameter requires that two arguments be passed instead of one.

If a list of media specified in an Export request contains media of more than one type, the request fails.

The Export command cannot be cancelled. Media can be "unmarked" for export via the ClearEject request.

A medium that is marked for export from the archive system cannot be reallocated to satisfy a client request, except to satisfy a query of the medium. Any other request (except ClearEject) received for that medium fails.

A medium can be exported even if it is currently allocated. Attempts to physically eject the medium fail until the medium is no longer in-use.

The total length of time the API software waits for a command status in asynchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on command requests submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for the Export command are set with `VSCMD_Export_SetDefaults`. If command-specific defaults are set for all commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Export command, the parameter identifier and the value to be used for the parameter can be submitted on the command itself.

The following fields can be retrieved from the status handle after a successful Export request:

- `VSID_ERROR_CODE`,
- `VSID_ERROR_CODE_ENTRY`,
- `VSID_ERROR_CODE_TABLE`,
- `VSID_MEDIA_ID`,
- `VSID_MEDIA_ID_ENTRY`,
- `VSID_MEDIA_ID_TABLE`,
- `VSID_SEQUENCE_NUM`,

- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Status_GetFields(l),
- VS_Initialize(l),
- VSCMD_Export_SetDefaults(l)

VSCMD_ Export_ SetDefaults

`VSCMD_Export_SetDefaults` sets command-level default parameters for the Export command.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

Synopsis

```
VST_BOOLEAN VSCMD_Export_SetDefaults
(
"…",
VSID_ENDFIELD )
```

Arguments

- `"…"` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_COMMENT (VST_COMMENT)	The export comment to use for these media. The comment appears with media on the archive's eject list.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for Export commands.
VSID_MEDIA_ID_LIST (int)	Number of media in the list.

Parameter Type	Description
(char **)	Array of media identifiers to export.
VSID_PRIORITY (VST_PRIORITY)	The execution priority (to override the default global execution priority) for Export commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Export commands. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE`.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in `USER_FIELD` for Export commands. `USER_FIELD` is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Export commands. Neither the API software nor VolServ uses `USER_FIELD`.

Return Values

VSCMD_Export_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.

Example

```
/*************************************
      *********
*
* FUNCTION: vst_export_defaults
*
* PURPOSE:
* This function sets the default
      parameters for the
* VSCMD_Export API call.
*
* PARAMETERS:
* none
*
*************************************
      *********/
#ifdef ANSI_C
   VST_BOOLEAN
      vst_export_defaults(void)
#else
   VST_BOOLEAN vst_export_defaults()
#endif
{
   VST_BOOLEAN          rc = VSE_FALSE;
   VST_PRIORITY         priority;
   VST_USER_FIELD       user_field;
   VST_TIME_OUT         timeout;
   VST_RETRY_LIMIT      retries;
   VST_STATUS_WAIT_FLAG wait_flag;
   VST_ENTERPRISE_ID    enterprise_id;
   VST_COMMENT          comment;

   /* get parameters from user */
   printf("*** Export default
      parameters ***\n" );
```

```
30    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
31    printf("\nEnter Export Comment ");
32    gets( comment);
33    /* set the default parameters */
34    rc = VSCMD_Export_SetDefaults(
35          VSID_PRIORITY,
         priority,
36          VSID_USER_FIELD,
         user_field,
37          VSID_TIMEOUT_VALUE,
         timeout,
38          VSID_RETRY_LIMIT,
         retries,
39          VSID_STATUS_WAIT_FLAG,
         wait_flag,
40          VSID_ENTERPRISE_ID,
         enterprise_id,
41          VSID_COMMENT,
         comment,
42          VSID_ENDFIELD);
43
44    return ( rc );
45 }
```

Notes

The `VSID_MEDIA_ID_LIST` parameter requires that two arguments be passed instead of one.

If a list of media specified in an Export request contains media of more than one type, the request fails.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Export(l)

VSCMD_ Import

`VSCMD_Import` is issued to request execution of VolServ Import requests.

A client uses Import requests to logically add media to the VolServ system. Upon receipt of an Import request, the specified media are added to the VolServ system. If a non-unique media identifier is specified, the Import for that medium fails.

Import is a logical operation. Media must be physically entered into an archive before they are available for client use (mounting,"…"). Entry is performed by an operator selecting the Enter function from the appropriate archive's console display. The `Enter` function is not available from the API.

Synopsis

```
VST_BOOLEAN VSCMD_Import (
VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD )
```

Arguments

- `handle` = The command handle for this Import request.
- `"…"` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The destination archive for the imported media. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_BATCH_NAME (VST_BATCH_NAME)	The batch name to be assigned to media that are automatically imported/checked in. Valid batch names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID	Identifier of the enterprise, if any, to receive final status on this request.
VSID_MANUFACTURER (VST_MANUFACTURER)	The manufacturer to be assigned to the imported media. Valid `VSID_MANUFACTURER` names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The MediaClass group where imported media is assigned.
VSID_MEDIA_ID_LIST (int)	Number of media in the list.
(char **)	Array of media identifiers to import.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_Import` returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- `VSE_FALSE` - The command failed. A return code of `VSE_FALSE` (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code's value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.
- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code's value is `VSE_VOLSERV`, the error occurred in VolServ, and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code's value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API, and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.

- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_import_execute
4  *
5  * PURPOSE:
6  * This function sends a import command
       to the
7  * VolServ software, prompting for all
       values needed.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN vst_import_execute(void)
15 #else
16    VST_BOOLEAN vst_import_execute()
17 #endif
18 {
19    VST_BOOLEAN          rc = VSE_FALSE;
20    int                  count;
21    char                 *
         medialist[VST_MAX_ITEMS];
22    VST_ARCHIVE_NAME     archive;
```

```
23    VST_MEDIA_CLASS_NAME mediaclass;
24    VST_BATCH_NAME       batch;
25    VST_MANUFACTURER_NAM manuf;
26    VST_COMMAND_HANDLE   cmd;
27
28    /* get parameters from user */
29    printf("*** Import parameters ***\n"
         );
30    printf("\nEnter Archive ");
31    gets( archive);
32    printf("\nEnter Media Class ");
33    gets( mediaclass);
34    printf("\nEnter Batch ");
35    gets( batch);
36    printf("\nEnter Manufacturer ");
37    gets( manuf);
38    count = vst_getmedialist(medialist,
         VST_MAX_ITEMS);
39    /* create the command handle */
40    /* Note that the command handle is
         not */
41    /* destroyed in this routine, but in
         */
42    /* vst_dispatch when final status is
         received. */
43    cmd = VS_Command_Create();
44    /* make sure that the command handle
         is not */
45    /* null. */
46    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
47    {
48       /* Send the command to the VolServ
         software. */
49       /* Note that status is not
         processed here. */
50       /* Instead, it is processed in the
         */
51       /* vst_dispatch routine. Also,
         note that */
52       /* default values such as timeout,
         value */
```

```
53       /* retry limit and priority are
         set as */
54       /* default parameters. */
55       rc = VSCMD_Import(cmd,
56          VSID_ARCHIVE_NAME,
         archive,
57          VSID_MEDIA_CLASS_NAME,
         mediaclass,
58          VSID_BATCH_NAME,        batch,
59          VSID_MANUFACTURER,      manuf,
60          VSID_MEDIA_ID_LIST,     count,
         medialist,
61          VSID_ENDFIELD);
62    }
63
64    return ( rc );
65 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ does not generate intermediate status in response to an Import request.

`VSCMD_Import` triggers MediaClass callbacks from VolServ.

The `VSID_MEDIA_ID_LIST` parameter requires that two arguments be passed instead of one.

If a list of media specified in an Import request contains media of more than one type, the request fails.

Import is a logical operation. Media must be physically entered into an archive by an operator before they are available for general use.

Media identifier values must be unique throughout a VolServ system. Non-unique names are rejected.

Media identifiers of media being imported into manual archives may contain alphanumeric and special characters, including spaces. However, spaces cannot be used as leading or trailing characters. If media in a manual archive can later be moved into an automated archive, the media identifiers must also conform to any naming restrictions imposed by the automated archive. For example, special characters may not be allowed in media identifiers in the automated archive.

Media type for the media is determined by the media type of the specified MediaClass group.

After the MediaClass capacity is reached, no additional media can be imported into the MediaClass group.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for Import requests is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on command requests submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.

- Command-specific parameter defaults for Import commands are set with `VSCMD_Import_SetDefaults`. If command-specific defaults are set for import commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Import command, the parameter identifier and the value to be used for the parameter can be submitted on the request itself.

The following fields can be retrieved from the status handle after a successful Import request:

- `VSID_ERROR_CODE`,
- `VSID_ERROR_CODE_ENTRY`,
- `VSID_ERROR_CODE_TABLE`,
- `VSID_MEDIA_ID`,
- `VSID_MEDIA_ID_ENTRY`,
- `VSID_MEDIA_ID_TABLE`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`, `V`
- `SID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_Import_SetDefaults(l)

VSCMD_ Import_ SetDefaults

`VSCMD_Import_SetDefaults` is the call issued to set the command default parameters for Import commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

Synopsis

```
VST_BOOLEAN VSCMD_Import_SetDefaults
(
"…",
VSID_ENDFIELD )
```

Arguments

- `"…"` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The destination archive (to override the default global destination archive) for the imported media. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.

Parameter Type	Description
VSID_BATCH_NAME (VST_BATCH_NAME)	The batch name to be assigned to media that are imported. Valid batch names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for Import commands.
VSID_MANUFACTURER (VST_MANUFACTURER)	The manufacturer to be assigned to the imported media. Valid manufacturer names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The MediaClass group where imported media are assigned.
VSID_MEDIA_ID_LIST (int)	Number of media in the list.
(char **)	Pointer to an array of media identifiers to import.
VSID_PRIORITY (VST_PRIORITY)	The execution priority for Import commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Import commands. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Import commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in status messages returned for Import commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Import_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_import_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Import API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_import_defaults(void)
15 #else
16    VST_BOOLEAN vst_import_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26    VST_BATCH_NAME          batch;
27    VST_MANUFACTURER_NAME   manuf;
28
29    /* get parameters from user */
30    printf("*** Import default
         parameters ***\n" );
31    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
32    printf("\nEnter Batch ");
33    gets( batch);
```

```
34    printf("\nEnter Manufacturer ");
35    gets( manuf);
36    /* set the default parameters */
37    rc = VSCMD_Import_SetDefaults(
38       VSID_PRIORITY,          priority,
39       VSID_USER_FIELD,
         user_field,
40       VSID_TIMEOUT_VALUE,     timeout,
41       VSID_RETRY_LIMIT,       retries,
42       VSID_STATUS_WAIT_FLAG,
         wait_flag,
43       VSID_ENTERPRISE_ID,
         enterprise_id,
44       VSID_BATCH_NAME,        batch,
45       VSID_MANUFACTURER,      manuf,
46       VSID_ENDFIELD);
47
48    return ( rc );
49 }
```

Notes

The `VSID_MEDIA_ID_LIST` parameter requires that two arguments be passed instead of one. The first argument passed is the entry number in the appropriate table. The second argument is a pointer to the location where the value is stored.

If a list of media specified in an Import request contains media of more than one type, the request fails.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Import(l)

VSCMD_ Intransit-Query

`VSCMD_IntransitQuery` is issued to request execution of VolServ Intransit Query commands.

A client uses Intransit Query requests to obtain information about media in the intransit state. The query returns a list of media identifiers.

A medium is considered to be intransit if it satisfies either of the following conditions:

- It is waiting to be entered into an archive as a result of Import, Mount, Move, Check-in, or a migration activity processing.
- It is in the homeless state.

Synopsis

```
VST_BOOLEAN VSCMD_IntransitQuery
( VST_COMMAND_HANDLE handle,
"...",
VSID_ENDFIELD )
```

Arguments

- `handle` = The command handle for this Intransit Query request.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status for this request.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status).Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE`.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_IntransitQuery returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.

- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_intransitquery_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_IntransitQuery
       API call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
```

```
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_intransitquery_execute(void)
14 #else
15    VST_BOOLEAN
         vst_intransitquery_execute()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_COMMAND_HANDLE      cmd;
20
21    printf("*** Intransit Query ***\n" );
22
23    /* create the command handle */
24    /* Note that the command handle is
         not */
25    /* destroyed in this routine, but in
         */
26    /* vst_dispatch when final status is
         received. */
27    cmd = VS_Command_Create();
28    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
29    {
30       /* Send the command to the VolServ
         software. */
31       /* Note that status is not
         processed here. */
32       /* Instead, it is processed in the
         */
33       /* vst_dispatch routine. Also,
         note that */
34       /* default values such as timeout,
         value */
35       /* retry limit and priority are
         set as */
36       /* default parameters. There are
         no */
37       /* command-specific parameters
         for */
38       /* intransit query. */
```

```
39        rc = VSCMD_IntransitQuery(cmd,
40
          VSID_ENDFIELD);
41    }
42    return ( rc );
43 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generate intermediate status in response to an Intransit Query request.

`VSCMD_IntransitQuery` does not trigger any MediaClass callbacks from VolServ.

The query option on the Intransit Query command is not currently supported by VolServ. VolServ unconditionally returns the identifiers of all media in the intransit state.

Only media in the `intransit` state are queried and returned.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on command requests submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for the Intransit Query request are set with `VSCMD_IntransitQuery_SetDefaults`. If command-specific defaults are set for Intransit Query commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Intransit Query request, the parameter identifier and the value to be used for the parameter can be submitted on the request itself.

The following fields can be retrieved from the status handle after a successful Intransit Query request:

- `VSID_MEDIA_ID`,
- `VSID_MEDIA_ID_ENTRY`,
- `VSID_MEDIA_ID_TABLE`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Command_GetFields(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Intransit_GetFields(l),
- VS_Status_GetFields(l),
- VS_Table_GetFields(l),
- VSCMD_IntransitQuery_SetDefaults(l)

VSCMD_ Intransit-Query_Set-Defaults

`VSCMD_IntransitQuery_SetDefaults` is issued to set command-level default parameters for Intransit Query commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

Synopsis

VST_BOOLEAN VSCMD_IntransitQuery_SetDefaults (
"…",
VSID_ENDFIELD)

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` =Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH	Name of the client dispatch routine to receive status for Intransit Query commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID	Identifier of the enterprise, if any, to receive final status for Intransit Query commands.

Parameter Type	Description
VSID_PRIORITY (VST_PRIORITY)	The execution priority (to override the default global execution priority) for Intransit Query commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive.The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Intransit Query commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Intransit Query commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Intransit Query commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_IntransitQuery_SetDefaults` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_intransitquery_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_IntransitQuery API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_intransitquery_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_intransitquery_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
```

```
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Intransit Query default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc =
         VSCMD_IntransitQuery_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_IntransitQuery(l)

VSCMD_Lock

`VSCMD_Lock` is issued to request execution of the VolServ Lock command.

The lock identifier assigned to the locked drive is returned to the client. This lock identifier must be used by clients on subsequent requests (such as Mount) for those drives.

A request to lock a drive that is busy (mounted or previously locked) queues until the drive becomes available.

A Lock request that specifies a drive pool or a list of drives should also indicate the number of drives from the pool/list to be locked. VolServ selects the drives to lock from within the pool/list according to drive availability.

A Lock request cannot specify a drive pool or a list of drives that spans archives.

A Lock request reserves one drive for exclusive use, if a quantity is not specified on the command.

VolServ considers only on-line drives as candidates to be locked. If there is not a sufficient number of on-line drives in the same archive to satisfy a Lock request, the Lock request fails.

If there is a sufficient number of on-line drives in the same archive to satisfy a Lock request, but the number of available on-line drives is not sufficient, the request waits until sufficient drives become available. Partial locks are not set.

Synopsis

```
VST_BOOLEAN VSCMD_Lock
( VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD )
```

Arguments

- `handle` = The command handle for this Lock request.
- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_DRIVE_COUNT (int)	Number of drives to lock.
VSID_DRIVE_EXCL_LIST (int)	Number of drives in the drive exclusion list.
(VST_DRIVE_ID *)	Pointer to the identifiers of the drives in the specified drive pool that are not to be locked.
VSID_DRIVE_ID (VST_DRIVE_ID)	Identifier of a single drive to be reserved for exclusive use.
VSID_DRIVE_ID_LIST (int)	Number of drives in list.
(VST_DRIVE_ID *)	Pointer to the identifiers of one or more drives to be reserved for exclusive use.
VSID_DRIVEPOOL_NAME (VST_DRIVEPOOL_NAME)	Name of the drive pool group to be reserved for exclusive use. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.

Parameter Type	Description
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status).Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE`.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in `USER_FIELD` for this request. `USER_FIELD` is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses `USER_FIELD`.

Return Values

VSCMD_Lock returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.

- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_lock_pool_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_Lock API call
       for drive
7  * pools.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_lock_pool_execute(void)
15 #else
16    VST_BOOLEAN vst_lock_pool_execute()
17 #endif
18 {
19    VST_BOOLEAN            rc =
         VSE_FALSE;
20    VST_DRIVE_ID
         excllist[VST_MAX_ITEMS];
21    int                    count;
```

```
22    VST_DRIVE_POOL_NAME     dp;
23    int                     drivecount;
24    VST_COMMAND_HANDLE      cmd;
25
26    printf("\nEnter Drive Pool Name
         ==>");
27    gets( dp );
28    printf("\nEnter drive exclusion
         list\n");
29    count =
         vst_getdrivelist(excllist,
         VST_MAX_ITEMS);
30    printf("Enter number of drives to
         lock ==> ");
31    drivecount = atoi(gets(input));
32
33    /* create the command handle */
34    /* Note that the command handle is
         not */
35    /* destroyed in this routine, but in
         */
36    /* vst_dispatch when final status is
         received. */
37    cmd = VS_Command_Create();
38    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
39    {
40       /* Send the command to the VolServ
         software. */
41       /* Note that status is not
         processed here. */
42       /* Instead, it is processed in the
         */
43       /* vst_dispatch routine. Also,
         note that */
44       /* default values such as
         timeout,*/
45       /* value retry limit and priority
         are set as */
46       /* default parameters. */
47       rc = VSCMD_Lock(cmd,
48                VSID_DRIVEPOOL_NAME, dp,
```

```
49                VSID_DRIVE_EXCL_LIST,
        count,excllist,
50                VSID_DRIVE_COUNT,
        drivecount,
51                VSID_ENDFIELD);
52    }
53    return(rc);
54 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generate intermediate status in response to a Lock request.

`VSCMD_Lock` does not trigger any MediaClass callbacks from VolServ.

The `VSID_DRIVE_ID_LIST` and `VSID_DRIVE_EXCEL_LIST` parameters require that two arguments be passed instead of one.

A Lock command that specifies a list of drives fails if the drives are not contained within the same physical archive.

A Lock command that specifies a drive pool that spans archives fails.

Any Mount or Dismount request containing the proper lock identifier has access to a locked drive.

If a Mount request does not specify a lock identifier for a locked drive, whether the drive is available for use or not, the Mount request waits until the drive is both unlocked and available.

If a Mount request specifies a drive pool and does not specify a lock identifier, only available and unlocked drives in the specified drive pool are considered to satisfy the Mount request. If there are no available, unlocked drives in the specified drive pool, the Mount request waits until a drive from the specified drive pool becomes available and unlocked.

There are three ways to specify drives for locking: by drive identifier, drive list, or drive pool (with or without the exclusion list).

A Lock command that is queued and awaiting resources can be cancelled via the Cancel command.

An Unlock command should be issued when the client no longer needs drives for exclusive use.

All parameters can be set for the specific request being sent by passing them to this function, or they can be set for all Lock requests using the `VSCMD_Lock_SetDefaults` function.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the API is not able to receive status for this request.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on command requests submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for the Lock command are set with `VSCMD_Lock_SetDefaults`. If command-specific defaults are set for the Lock command, they override the global defaults for all Lock requests.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Lock request, the parameter identifier and the value to be used for the parameter can be submitted on the command request itself.

The following fields can be retrieved from the status handle after a successful Lock request:

- VSID_Drive_ID,
- VSID_DRIVE_ID_ENTRY,
- `VSID_DRIVE_ID_TABLE`,
- VSID_ERROR_CODE,
- VSID_ERROR_CODE_ENTRY,
- `VSID_ERROR_TABLE`,
- `VSID_LOCK_ID`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_Lock_SetDefaults(l),
- VSCMD_Unlock(l)

VSCMD_Lock_SetDefaults

VSCMD_Lock_SetDefaults is issued to set the command-level default parameters for the Lock command.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

Synopsis

```
VST_BOOLEAN VSCMD_Lock_SetDefaults
(
"…",
VSID_ENDFIELD )
```

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_DRIVE_COUNT (int)	Number of drives to lock.
VSID_DRIVE_EXCL_LIST (int)	Number of drives in list.
(VST_DRIVE_ID *)	Pointer to a list of drives to exclude from the drive pool.
VSID_DRIVE_ID (VST_DRIVE_ID)	Drive identifier of drive to lock.

Parameter Type	Description
VSID_DRIVE_ID_LIST (int)	Number of drives in list.
(VST_DRIVE_ID *)	Pointer to a list of drives to lock.
VSID_DRIVEPOOL_NAME (VST_DRIVEPOOL_NAME)	The drive pool of drives to lock. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for Lock commands.
VSID_PRIORITY (VST_PRIORITY)	The execution priority for Lock commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software (to override the default global retry limit) for Lock commands. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status).Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE`.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Lock commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Lock commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Lock_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_lock_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Lock API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN vst_lock_defaults(void)
15 #else
16    VST_BOOLEAN vst_lock_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Lock default parameters
         ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_Lock_SetDefaults(
32          VSID_PRIORITY,
         priority,
```

```
33              VSID_USER_FIELD,
          user_field,
34              VSID_TIMEOUT_VALUE,
          timeout,
35              VSID_RETRY_LIMIT,
          retries,
36              VSID_STATUS_WAIT_FLAG,
          wait_flag,
37              VSID_ENTERPRISE_ID,
          enterprise_id,
38              VSID_ENDFIELD);
39     return ( rc );
40 }
```

Notes

The `VSID_DRIVE_ID_LIST` and `VSID_DRIVE_EXCL_LIST` parameters require that two arguments be passed instead of one. The first argument passed is the entry number in the appropriate table. The second argument is a pointer to the location where the value is stored.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Lock(l)

VSCMD_ MediaClass-Query

`VSCMD_MediaClassQuery` is issued to request execution of VolServ Media Class Query commands.

Media Class Query commands are used to obtain information about MediaClass groups in the VolServ system.

Upon receipt of a Media Class Query command, where obtains the requested information about the specified MediaClass groups and returns this information to the client.

Synopsis

VST_BOOLEAN VSCMD_MediaClassQuery
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)

Arguments

- `handle` = The command handle for this Media Class Query request.
- `“...”` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLASS_QUERY_OPT (VST_QUERY_LIST_OPTION)	Indicates the amount of media information being requested for each medium in each reported MediaClass group. Valid `VSID_CLASS_QUERY_OPT` values are enumerated in the *vs_types.h* file.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	If information is being requested on a single MediaClass group, specifies the name of that MediaClass group. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is being requested for a single specified MediaClass group or on all MediaClass groups. Valid `VSID_QUERY_OPTION` values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status).Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_MediaClassQuery returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.

  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code's value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.

- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code's value is `VSE_VOLSERV`, the error occurred in VolServ, and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code's value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API, and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NOTINITIALIZED` - The VolServ API is not initialized.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.
- `VSE_ERR_SEND` - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediaclassquery_execute
4  *
5  * PURPOSE:
6  * This executes the
       VSCMD_MediaClassQuery API call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_mediaclassquery_execute(void)
14 #else
15    VST_BOOLEAN
         vst_mediaclassquery_execute()
16 #endif
17 {
18    VST_BOOLEAN rc = VSE_FALSE;
19    VST_QUERY_OPTION queryopt;
20    VST_QUERY_LIST_OPTION querylistopt;
21    VST_MEDIA_CLASS_NAME mediaclass;
22    VST_COMMAND_HANDLE cmd;
23
24    /* get parameters from user */
25    printf("*** MediaClass Query
         parameters ***\n" );
26    printf("0) Query by Media Class Name,
         1) Query all ==> " );
27    queryopt = atoi(gets(input));
28
29    if (queryopt == 0)
30    {
31       printf("\nEnter Media Class Name
         ==>");
32       gets( mediaclass);
33    }
```

```
34    printf("\n0) no media list, 1) Media
         List, 2) Media List Details ==>
         ");
35    querylistopt = atoi(gets(input));
36
37    /* create the command handle */
38    /* Note that the command handle is
         not */
39    /* destroyed in this routine, but in
         */
40    /* vst_dispatch when final status is
         received. */
41    cmd = VS_Command_Create();
42    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
43    {
44       /* Send the command to the VolServ
         software. */
45       /* Note that status is not
         processed here. */
46       /* Instead, it is processed in the
         */
47       /* vst_dispatch routine. Also,
         note that */
48       /* default values such as
         timeout,*/
49       /* value retry limit and priority
         are set as */
50       /* default parameters. */
51       if (queryopt == 0)
52       {
53          /* query one media class */
54          rc =
         VSCMD_MediaClassQuery(cmd,
55                VSID_QRY_OPTION,
         queryopt,
56                VSID_CLASS_QRY_OPTION,
         querylistopt,
57                VSID_MEDIA_CLASS_NAME,
         mediaclass,
58                VSID_ENDFIELD);
59       }
```

```
60       else
61       {
62          /* query all media classes */
63          rc =
         VSCMD_MediaClassQuery(cmd,
64                      VSID_QRY_OPTION,
         queryopt,
65
         VSID_CLASS_QRY_OPTION,
         querylistopt,
66                      VSID_ENDFIELD);
67       }
68    }
69    return ( rc );
70 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generate intermediate status in response to a MediaClassQuery request.

`VSCMD_MediaClassQuery` does not trigger any MediaClass callbacks from VolServ.

When all media classes are requested, each MediaClass status is reported in a group of one or more intermediate status messages.

If a request for MediaClass status determines there is no MediaClass status on which to report, the status message returns a `STATUS_FAIL` with error of `ERR_NOTFOUND`.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive intermediate and final status for Media Class Query requests submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—.global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.
- Command-specific parameter defaults for Media Class Query commands are set with VSCMD_MediaClassQuery_SetDefaults. If command-specific defaults are set for the Media Class Query commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Media Class Query command, the parameter identifier and the value to be used for the parameter can be submitted for the specific command itself.

The following fields can be retrieved from the status handle after a successful Media Class Query request:

- VSID_MEDIACLASS_HANDLE,
- VSID_MEDIACLASS_HANDLE_ENTRY,
- VSID_MEDIACLASS_HANDLE_TABLE,

- VSID_QUERY_OPTION,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Command_GetFields(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_MediaClass_GetFields(l),
- VS_Status_GetFields(l),
- VS_Table_GetFields(l),
- VSCMD_MediaClassQuery_SetDefaults(l)

VSCMD_ MediaClass-Query_ SetDefaults

VSCMD_MediaClassQuery_SetDefaults is issued to set the command-level default parameters for the Media Class Query command.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

Synopsis

VST_BOOLEAN VSCMD_MediaClassQuery_SetDefaults (
"…",
VSID_ENDFIELD)

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD =Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLASS_QUERY_OPT (VST_QUERY_LIST_OPTION)	Indicates the amount of media information being requested for each medium in each reported MediaClass group. Valid VSID_CLASS_QUERY_OPT values are enumerated in the *vs_types.h* file.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for Media Class Query commands.

Parameter Type	Description
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status for Media Class Query commands.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	If information is being requested on a single MediaClass group, specifies the name of that MediaClass group. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_PRIORITY (VST_PRIORITY)	The execution priority for Media Class Query commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is being requested for a single specified MediaClass group or on all MediaClass groups. Valid `VSID_QUERY_OPTION values are enumerated in the vs_types.h file.`
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Media Class Query commands. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE`.

Parameter Type	Description
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Media Class Query commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Media Class Query commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_MediaClassQuery_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /****************************************
       *********
2  *
3  * FUNCTION:
       vst_mediaclassquery_defaults
4  *
5  * PURPOSE:
```

```
6  * This function sets the default
       parameters for the
7  * VSCMD_MediaClassQuery API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 ************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
       vst_mediaclassquery_defaults(void
       )
15 #else
16    VST_BOOLEAN
       vst_mediaclassquery_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
       VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
       enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Media Class Query default
       parameters ***\n" );
29    vst_promptforglobals(&priority,
       user_field, &timeout, &retries,
       &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc =
       VSCMD_MediaClassQuery_SetDefaults
       (
32           VSID_PRIORITY,
       priority,
33           VSID_USER_FIELD,
       user_field,
```

```
34              VSID_TIMEOUT_VALUE,
          timeout,
35              VSID_RETRY_LIMIT,
          retries,
36              VSID_STATUS_WAIT_FLAG,
          wait_flag,
37              VSID_ENTERPRISE_ID,
          enterprise_id,
38              VSID_ENDFIELD);
39     return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_MediaClassQuery(l)

VSCMD_ MediaQuery

VSCMD_MediaQuery is issued to request execution of the VolServ Media Query command.

Upon receipt of a Media Query command, VolServ obtains the requested information about the specified media and returns this information to the client.

Synopsis

```
VST_BOOLEAN VSCMD_MediaQuery
( VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD )
```

Arguments

- "..." = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.
VSID_MEDIA_ID_LIST (int)	Number of media in the list.
(char **)	List of media identifiers to query.

Parameter Type	Description
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is being requested for each medium specified in a list of one or more media or whether information is being requested for all media. Valid `VSID_QUERY_OPTION` values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status).Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE`.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_MediaQuery returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.

- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediaquery_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_MediaQuery API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
```

```
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_mediaquery_execute(void)
14 #else
15    VST_BOOLEAN vst_mediaquery_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_QUERY_OPTION     queryopt;
20    int                  count;
21    char                 *
         medialist[VST_MAX_ITEMS];
22    VST_COMMAND_HANDLE   cmd;
23
24    /* get parameters from user */
25    printf("*** Media Query parameters
         ***\n" );
26    printf("0) Query by media list, 1)
         Query all ==> " );
27    queryopt = atoi(gets(input));
28
29    if (queryopt == 0)
30    {
31       count =
         vst_getmedialist(medialist,
         VST_MAX_ITEMS);
32    }
33
34    /* create the command handle */
35    /* Note that the command handle is
         not */
36    /* destroyed in this routine, but in
         */
37    /* vst_dispatch when final status is
         received. */
38    cmd = VS_Command_Create();
39    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
40    {
41       /* Send the command to the VolServ
         software. */
```

```
42       /* Note that status is not
         processed here. */
43       /* Instead, it is processed in the
         */
44       /* vst_dispatch routine. Also,
         note that */
45       /* default values such as
         timeout,*/
46       /* value retry limit and priority
         are set as */
47       /* default parameters. */
48       rc = VSCMD_MediaQuery(cmd,
49             VSID_QRY_OPTION, queryopt,
50             VSID_MEDIA_ID_LIST, count,
         medialist,
51             VSID_ENDFIELD);
52    }
53    return ( rc );
54 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generate intermediate status in response to a Media Query request.

`VSCMD_MediaQuery` does not trigger any MediaClass callbacks from VolServ.

The `VSID_MEDIA_ID_LIST` parameter requires that two arguments be passed instead of one.

When information is requested for more than one media, the grouped information is returned in one or more intermediate status messages.

A Media Query can query any media in the VolServ system. Media specified in a single Media Query request are not required to be located in the same archive.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status for Media Query requests submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Media Query commands are set with `VSCMD_MediaQuery_SetDefaults`. If command-specific defaults are set for the Media Query commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Media Query command, the parameter identifier and the value to be used for the parameter can be submitted for the specific command itself.

The following fields can be retrieved from the Status handle after a successful Media Query request:

- VSID_ERROR_CODE,
- VSID_ERROR_CODE_ENTRY,
- VISD_ERROR_CODE_TABLE,
- VSID_MEDIA_HANDLE,
- VSID_MEDIA_HANDLE_ENTRY,
- VSID_MEDIA_HANDLE_TABLE,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Command_GetFields(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Media_GetFields(l),
- VS_Status_GetFields(l),
- VS_Table_GetFields(l),
- VSCMD_MediaQuery_SetDefaults(l)

VSCMD_ MediaQuery_ SetDefaults

`VSCMD_MediaQuery_SetDefaults` sets the command-level default parameters for the Media Query commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

Synopsis

```
VST_BOOLEAN VSCMD_MediaQuery_SetDefaults
(
"…",
VSID_ENDFIELD )
```

Arguments

- `"..."` = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for Media Query commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Media Query commands.
VSID_MEDIA_ID_LIST (int)	Number of media identified in the list.
(char **)	List of media identifier to query.

Parameter Type	Description
VSID_PRIORITY (VST_PRIORITY)	The execution priority for Media Query commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is being requested for each medium specified in a list of one or more media or whether information is being requested for all media. Valid VSID_QUERY_OPTION values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Media Query commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Media Query commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Media Query commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_MediaQuery_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediaquery_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_MediaQuery API call.
8  *
9  * PARAMETERS:
10 * none
11 *
```

```
12 ************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_mediaquery_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_mediaquery_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Media Query default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_MediaQuery_SetDefaults(
32          VSID_PRIORITY,
         priority,
33          VSID_USER_FIELD,
         user_field,
34          VSID_TIMEOUT_VALUE,
         timeout,
35          VSID_RETRY_LIMIT,
         retries,
36          VSID_STATUS_WAIT_FLAG,
         wait_flag,
37          VSID_ENTERPRISE_ID,
         enterprise_id,
38          VSID_ENDFIELD);
39    return ( rc );
```

```
40 }
```

Notes

The `VSID_MEDIA_ID_LIST` parameter requires that two arguments be passed instead of one. The first argument passed is the entry number in the appropriate table. The second argument is a pointer to the location where the value is stored.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_MediaQuery(l)

VSCMD_ MediaType-Query

VSCMD_MediaTypeQuery is issued to request execution of the VolServ Media Type Query request.

The Media Type Query request is used to obtain information about media types defined in the VolServ system.

Upon receipt of a Media Type Query request, VolServ obtains the requested information about the specified media type and returns this information to the client.

Synopsis

VST_BOOLEAN VSCMD_MediaTypeQuery
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on this request.

Parameter Type	Description
VSID_MEDIA_TYPE_LIST (int)	If VSE_QUERY_OPTION_NONE was specified for VSID_QUERY_OPTION, indicates the number of media types specified in the Media Type list. The maximum number of media types that can be specified is 32.
(char **)	If VSE_QUERY_OPTION_NONE was specified for VSID_QUERY_OPTION, specifies the names of the media types for which information is being requested. Valid Media Type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is being requested for each media type specified in a list of one or more media types or whether information is being requested for all media types. Valid VSID_QUERY_OPTION values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_MediaTypeQuery returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.

  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code's value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.

- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code's value is `VSE_VOLSERV`, the error occurred in VolServ, and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code's value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API, and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NOTINITIALIZED` - The VolServ API is not initialized.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediatypequery_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_MediaTypeQuery
       API call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_mediatypequery_execute(void)
14 #else
15    VST_BOOLEAN
         vst_mediatypequery_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_BOOLEAN          done =
         VSE_FALSE;
20    VST_QUERY_OPTION     queryopt;
21    int                  /count = 0;
22    char                 *
         temp_media_type;
23    char                 *
         mediatypelist[VST_MAX_ITEMS];
24    VST_COMMAND_HANDLE cmd;
25
26    /* get parameters from user */
```

```
27    printf("*** Media Type Query
         parameters ***\n" );
28    printf("0) Query by media type list,
         1) Query all ==> " );
29    queryopt = atoi(gets(input));
30
31    if (queryopt == 0)
32    {
33       count =
         vst_getmediatypelist(mediatypelis
         t, VST_MAX_ITEMS);
34    }
35
36    /* create the command handle */
37    /* Note that the command handle is
         not */
38    /* destroyed in this routine, but in
         */
39    /* vst_dispatch when final status is
         received. */
40    cmd = VS_Command_Create();
41    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
42    {
43       /* Send the command to the VolServ
         software. */
44       /* Note that status is not
         processed here. */
45       /* Instead, it is processed in the
         */
46       /* vst_dispatch routine. Also,
         note that */
47       /* default values such as
         timeout,*/
48       /* value retry limit and priority
         are set as */
49       /* default parameters. */
50       if (queryopt == 0)
51       {
52          /* query a list of media types
         */
53          rc = VSCMD_MediaTypeQuery(cmd,
```

```
54              VSID_QRY_OPTION,
        queryopt,
55              VSID_MEDIA_TYPE_LIST, count,
        mediatypelist,
56              VSID_ENDFIELD);
57        }
58        else
59        {
60           /* query all media types */
61           rc = VSCMD_MediaTypeQuery(cmd,
62
        VSID_QRY_OPTION, queryopt,
63                           VSID_ENDFIELD);
64        }
65     }
66     else
67     {
68        rc = VSE_FALSE;
69     }
70     return ( rc );
71  }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generate intermediate status in response to a Media Type Query request.

Media Type Query status are cumulative. Each status is added to the previous status; therefore, after the final status, the status handle contains all desired information.

`VSCMD_MediaTypeQuery` does not trigger any MediaClass callbacks from VolServ.

The `VSID_MEDIA_TYPE_LIST` parameter requires that two arguments be passed instead of one.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status for Media Type Query requests submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Media Type Query commands are set with `VSCMD_MediaQuery_SetDefaults`. If command-specific defaults are set for the Media Type Query commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Media Type Query command, the parameter identifier and the value to be used for the parameter can be submitted for the specific command itself.

The following fields can be retrieved from the status handle after a successful Media Type Query request:

- VSID_ERROR_CODE,
- VSID_ERROR_CODE_ENTRY,
- VSID_ERROR_CODE_TABLE,
- VSID_MEDIATYPE_HANDLE,
- VSID_MEDIATYPE_HANDLE_ENTRY,
- VSID_MEDIATYPE_HANDLE_TABLE,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Command_GetFields(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Media_GetFields(l),
- VS_Status_GetFields(l),
- VS_Table_GetFields(l),
- VSCMD_MediaTypeQuery_SetDefaults(l)

VSCMD_ MediaType-Query_ SetDefaults

`VSCMD_MediaTypeQuery_SetDefaults` is issued to set the command-level default parameters for Media Type Query commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

Synopsis

VST_BOOLEAN VSCMD_MediaTypeQuery_SetDefaults (
"…",
VSID_ENDFIELD)

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for Media Type Query commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Media Type Query commands.
VSID_MEDIA_TYPE_LIST (int)	If `VSE_QUERY_OPTION_NONE` was specified for `VSID_QUERY_OPTION`, indicates the number of media types specified in the list of media types.
(char **)	If `VSE_QUERY_OPTION_NONE` was specified for `VSID_QUERY_OPTION`, specifies the names of the media types for which information is being requested. Valid Media Type Query names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_PRIORITY (VST_PRIORITY)	The execution priority for Media Type Query commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicate whether information is being requested for each media type specified in a list of one or more media types or whether information is being requested for all media types. Valid `VSID_QUERY_OPTION` values are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Media Type Query commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Media Type Query commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Media Type Query commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_MediaTypeQuery_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.

- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mediatypequery_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_MediaTypeQuery API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_mediatypequery_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_mediatypequery_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
```

```
26
27    /* get parameters from user */
28    printf("*** Modify Pool default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc =
         VSCMD_MediaTypeQuery_SetDefaults(
32          VSID_PRIORITY,
         priority,
33          VSID_USER_FIELD,
         user_field,
34          VSID_TIMEOUT_VALUE,
         timeout,
35          VSID_RETRY_LIMIT,
         retries,
36          VSID_STATUS_WAIT_FLAG,
         wait_flag,
37          VSID_ENTERPRISE_ID,
         enterprise_id,
38          VSID_ENDFIELD);
39    return ( rc );
40 }
```


Notes

The `VSID_MEDIA_TYPE_LIST` parameters requires that two arguments be passed instead of one.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),

- VSCMD_MediaTypeQuery(l)

VSCMD_ ModifyMedia

A client uses Modify Media requests to create or modify the attribute values of an existing medium.

Synopsis

```
VST_BOOLEAN VSCMD_ModifyMedia
(
"…",
VSID_ENDFIELD )
```

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_BATCH_NAME (VST_BATCH_NAME)	The batch name of the medium. A valid batch name may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on all requests.
VSID_FIELD_LIST (int)	Number of field identifiers in the list.

Parameter Type	Description
(VST_COUNT *)	Pointer to an array of field identifiers to associate with the user statistics.
VSID_MANUFACTURER (VST_MANUFACTURER_NAME)	The manufacturer to be assigned to imported medium. Valid manufacturer names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MEDIA_ID (VST_MEDIA_ID)	Media identifier of the medium to update.
VSID_MEDIA_STAT_VALUE_LIST (int)	Number of user statistics in the list.
(char **)	An array of user statistics to associate with the medium.
VSID_MEDIA_STAT_OPTION_LIST (int)	Number of media statistic options in the list.
(VST_MEDIA_STAT_OPTION *)	An array of media statistic options to place on the list of user statistics. Valid `VSID_MEDIA_STAT_OPTION_LIST` values are enumerated in the *vs_types.h* file.
VSID_MODMEDIA_OPTION (VST_MODMEDIA_OPTION)	The option for the medium's user statistics. Valid `VSID_MODMEDIA_OPTION` values are enumerated in the *vs_types.h* file.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status).Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VS_MediaType_SetFields returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed. A return code of VSE_FALSE (which is 0) means the command failed.

  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code's value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.

- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code's value is `VSE_VOLSERV`, the error occurred in VolServ, and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code's value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API, and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NOTINITIALIZED` - The VolServ API is not initialized.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.
- `VSE_ERR_SEND` - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_modifymedia_execute
4  *
5  * PURPOSE:
6  * This function actually tests the
       VSCMD_ModifyMedia
7  * API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_modifymedia_execute(void)
15 #else
16    VST_BOOLEAN
         vst_modifymedia_execute()
17 #endif
18 {
19    int                    i;
20    int                    num;
21    VST_BOOLEAN            rc =
         VSE_FALSE;
22    VST_COMMAND_HANDLE     cmdh;
23    VST_MEDIA_ID           mediaid;
24    VST_BATCH_NAME         batch;
25    VST_MANUFACTURER_NAME
         manufacturer;
26    VST_MODMEDIA_OPTION    modopt;
27    VST_COUNT
         field[VSD_MAX_MEDIA_STATS];
28    VST_MEDIA_STAT_OPTION
         option[VSD_MAX_MEDIA_STATS];
29    char
          * value[VSD_MAX_MEDIA_STATS];
30    char
         valuearray[VSD_MAX_MEDIA_STATS][V
         SD_MEDIA_STAT_VALUE_LEN];
```

```
31
32    /* get parameters from user */
33    printf("*** Modify Media Parameters
         ***\n");
34    printf ( "Enter the media id to
         modify ==> " );
35    gets(mediaid);
36    printf ( "Enter the batch name
         (return for none) ==> " );
37    gets(batch);
38    printf ( "Enter the manufacturer name
         ==> " );
39    gets(manufacturer);
40    printf ( "Enter the modify media
         option
         (1-Delete/2-Update/3-Replace)==>
         " );
41    modopt = (VST_MODMEDIA_OPTION)
         atoi(gets(input));
42    printf ( "Enter the number of user
         statistics ==> " );
43    num = atoi(gets(input));
44
45    /* loop through the number of user
         statistics */
46    for ( i = 0 ; i < num && i <
         VSD_MAX_MEDIA_STATS ; i++ )
47    {
48       printf ( "Enter field index[%d]
         ==> ", i );
49       field[i] = (VST_COUNT)
         atoi(gets(input));
50       printf ( "Enter user
         statistic[%d] ==> ", i );
51        gets(valuearray[i]);
52       value[i] = valuearray[i];
53       printf ( "Enter user stat
         option[%d] (1-delete/2-update)
         ==>", i );
54       option[i] =
         (VST_MEDIA_STAT_OPTION)
         atoi(gets(input));
```

```
55    }
56
57    /* Create the command (assume that
         the api is */
58    /* initialized). Note that status is
         processed */
59    /* in the vst_dispatch routine. */
60    /* Also, the command handle created
         here is */
61    /* destroyed in the dispatch routine.
         */
62    cmdh = VS_Command_Create();
63    if (cmdh != (VST_COMMAND_HANDLE)
         NULL)
64    {
65    /* execute the modify media command
         */
66    /* common parameters such as
         priority, timeout */
67    /* value, etc, have been set through
         the */
68    /* VS_Global_SetFields or */
69    /* VSCMD_ModifyMedia_SetDefaults. */
70    rc = VSCMD_ModifyMedia ( cmdh,
71          VSID_MEDIA_ID,
         mediaid,
72          VSID_BATCH_NAME,
         batch,
73          VSID_MANUFACTURER,
         manufacturer,
74          VSID_MODMEDIA_OPTION,
         modopt,
75          VSID_FIELD_LIST,
         num, field,
76          VSID_MEDIA_STAT_VALUE_LIST,
         num, value,
77          VSID_MEDIA_STAT_OPTION_LIST,
         num,option,
78          VSID_ENDFIELD );
79    }
80
81    return ( rc );
```

```
82 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ does not generate intermediate status in response to a Modify Media request.

`VSCMD_ModifyMedia` does not trigger any MediaClass callbacks from VolServ.

The `VSID_FIELD_LIST`, `VSID_MEDIA_STAT_OPTION_LIST`, and `VSID_MEDIA_STAT_VALUE_LIST` parameters require that two arguments be passed instead of one.

Number of items in the field, media stat, and media stat option lists must be equal.

If the `VSID_MODMEDIA_OPTION` is set to `VSE_MODMEDIA_OPTION_REPLACE`, the current medium statistics are deleted, and the statistics given in the `VSID_MEDIA_STAT_VALUE_LIST` are added for the medium.

If the `VSID_MODMEDIA_OPTION` is set to `VSE_MODMEDIA_OPTION_DELETE`, the field, stat option, and stat value lists are ignored, and all current medium statistics are deleted.

A medium may have as many field values as desired. However, only up to 16 can be updated at one time through the Modify Media command.

All medium statistics are kept in character format. If numeric values are to be kept, they should be left-filled with zeros for proper comparisons.

When a medium is exported, its statistics are deleted after the medium is ejected from the system.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status for Modify Media requests submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Modify Media commands are set with `VSCMD_ModifyMedia_SetDefaults`. If command-specific defaults are set for the Modify Media commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Modify Media command, the parameter identifier and the value to be used for the parameter can be submitted for the specific command itself.

The following fields can be retrieved from the status handle after a successful Modify Media request:

- VSID_ERROR_CODE,
- VSID_ERROR_CODE_ENTRY,
- VSID_ERROR_CODE_TABLE,
- VSID_FIELD,
- VSID_FIELD_ENTRY,
- VSID_FIELD_TABLE,
- VSID_MEDIA_ID,
- VSID_MEDIA_ID_ENTRY,
- VSID_MEDIA_ID_TABLE,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- VS_Global_SetFields(l),
- VSCMD_ModifyMedia_SetDefaults(l)

VSCMD_ ModifyMedia_ SetDefaults

`VSCMD_ModifyMedia_SetDefaults` sets command-level default parameters for all `VSCMD_ModifyMedia` commands.

These defaults override those set in `VS_Global_SetFields`. Also, the values set here can be overridden by passing parameters to `VSCMD_ModifyMedia`.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

Synopsis

```
VST_BOOLEAN VSCMD_ModifyMedia_SetDefaults
(
"…",
 VSID_ENDFIELD )
```

Arguments

- **"…"** = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_BATCH_NAME (VST_BATCH_NAME)	The batch name of the medium. A valid batch name may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for Modify Media commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for Modify Media commands.
VSID_FIELD_LIST (int)	Number of field identifiers in the list.
(VST_COUNT *)	Pointer to an array of field identifiers to associate with the user statistics.
VSID_MANUFACTURER (VST_MANUFACTURER_NAME)	The manufacturer to be assigned to imported medium. Valid manufacturer names may contain up to 32 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MEDIA_ID (VST_MEDIA_ID)	Media identifier of the medium to `modify.`
VSID_MEDIA_STAT_OPTION_LIST (int)	Number of items in the list.
(VST_MEDIA_STAT_OPTION *)	An array of media statistic options to place on the list of user statistics.
VSID_MEDIA_STAT_VALUE_LIST (int)	Number of user statistics in the list.
(char **)	An array of user statistics to associate with the medium.
VSID_MODMEDIA_OPTION (VST_MODMEDIA_OPTION)	The option for the medium's user statistics. Valid `VSID_MODMEDIA_OPTION values are enumerated in the vs_types.h file.`

Parameter Type	Description
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Modify Media commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Modify Media commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG(VST_STATUS_ WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for a command. Valid options are VSE_TRUE (API waits for final status) and VSE_FALSE (API does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Modify Media commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Modify Media commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_ModifyMedia returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_modifymedia_execute
4  *
5  * PURPOSE:
6  * This function actually tests the
       VSCMD_ModifyMedia
7  * API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 **************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_modifymedia_execute(void)
15 #else
16    VST_BOOLEAN
         vst_modifymedia_execute()
17 #endif
18 {
19    int                    i;
20    int                    num;
```

```
21    VST_BOOLEAN             rc =
         VSE_FALSE;
22    VST_COMMAND_HANDLE      cmdh;
23    VST_MEDIA_ID            mediaid;
24    VST_BATCH_NAME          batch;
25    VST_MANUFACTURER_NAME
         manufacturer;
26    VST_MODMEDIA_OPTION     modopt;
27    VST_COUNT
         field[VSD_MAX_MEDIA_STATS];
28    VST_MEDIA_STAT_OPTION
         option[VSD_MAX_MEDIA_STATS];
29    char
         value[VSD_MAX_MEDIA_STATS];
30    char
         valuearray[VSD_MAX_MEDIA_STATS]

         [VSD_MEDIA_STAT_VALUE_LEN];
31
32    /* get parameters from user */
33    printf("*** Modify Media Parameters
         ***\n");
34    printf ( "Enter the media id to
         modify ==> " );
35    gets(mediaid);
36    printf ( "Enter the batch name
         (return for none) ==> " );
37    gets(batch);
38    printf ( "Enter the manufacturer name
         ==> " );
39    gets(manufacturer);
40    printf ( "Enter the modify media
         option
         (1-Delete/2-Update/3-Replace)==>
         " );
41    modopt = (VST_MODMEDIA_OPTION)
         atoi(gets(input));
42    printf ( "Enter the number of user
         statistics ==> " );
43    num = atoi(gets(input));
44
```

```
45    /* loop through the number of user
         statistics */
46    for ( i = 0 ; i < num && i <
         VSD_MAX_MEDIA_STATS ; i++ )
47    {
48       printf ( “Enter field index[%d]
         ==> “, i );
49       field[i] = (VST_COUNT)
         atoi(gets(input));
50       printf ( “Enter user
         statistic[%d] ==> “, i );
51        gets(valuearray[i]);
52       value[i] = valuearray[i];
53       printf ( “Enter user stat
         option[%d] (1-delete/2-update)
         ==>“, i );
54       option[i] =
         (VST_MEDIA_STAT_OPTION)
         atoi(gets(input));
55    }
56
57    /* Create the command (assume that
         the api is */
58    /* initialized). Note that status is
         processed */
59    /* in the vst_dispatch routine. */
60    /* Also, the command handle created
         here is */
61    /* destroyed in the dispatch routine.
         */
62    cmdh = VS_Command_Create();
63    if (cmdh != (VST_COMMAND_HANDLE)
         NULL)
64    {
65    /* execute the modify media command
         */
66    /* common parameters such as
         priority, timeout */
67    /* value, etc, have been set through
         the */
68    /* VS_Global_SetFields or */
69    /* VSCMD_ModifyMedia_SetDefaults. */
```

```
70    rc = VSCMD_ModifyMedia ( cmdh,
71       VSID_MEDIA_ID,
         mediaid,
72       VSID_BATCH_NAME,
         batch,
73       VSID_MANUFACTURER,
         manufacturer,
74       VSID_MODMEDIA_OPTION,
         modopt,
75       VSID_FIELD_LIST,
         num, field,
76       VSID_MEDIA_STAT_VALUE_LIST,
         num, value,
77       VSID_MEDIA_STAT_OPTION_LIST,
         num, option,
78       VSID_ENDFIELD );
79    }
80
81    return ( rc );
82 }
```


Notes

The `VSID_FIELD_LIST`, `VSID_MEDIA_STAT_OPTION_LIST`, and `VSID_MEDIA_STAT_VALUE_LIST` parameters require that two arguments be passed instead of one.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- VS_Global_SetFields(l),
- VSCMD_ModifyMedia(l)

VSCMD_ CreatePool

VSCMD_CreatePool creates a new VolServ drive pool. The drive members of the drive pools are specified in the command. Drive pools are non-exclusive; drives may exist in more than one pool. Drive pools allow logical groupings of system drives for simplified reference when a specific drive does not need to be specified in a command, such as the Mount command.

After receiving a VSCMD_CreatePool request, VolServ creates a new drive pool and returns status to the client which indicates the success or failure of the request.

Synopsis

```
VST_BOOLEAN VSCMD_CreatePool
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)
```

Arguments

- handle = The command handle for the Create Pool request.
- “…” = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	Name of the drive pool to be created. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_DRIVE_ID_LIST (int)	Number of drives to include in the new drive pool.
(VST_DRIVE_ID *)	Pointer to the list of drives to be included in the new drive pool.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on this request.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_CreatePool returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.

  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code value is `VSE_VOLSERV`, the error occurred in VolServ and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API software and the client uses `VST_ERROR_CODE` to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_createpool_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_CreatePool API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_createpool_execute(void)
14 #else
15    VST_BOOLEAN vst_createpool_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_DRIVE_POOL_NAME  dp;
20    int                  count;
21    VST_DRIVE_ID
         drivelist[VST_MAX_ITEMS];
22    VST_COMMAND_HANDLE   cmd;
23
24    /* get parameters from user */
25    printf("*** Create Pool Parameters
         ***\n" );
26    printf("\nEnter Drive Pool Name
         ==>");
27    gets( dp );
28    count = vst_getdrivelist(drivelist,
         VST_MAX_ITEMS);
29
30    /* create the command handle */
31    /* Note that the command handle is
         not */
32    /* destroyed in this routine, but in
         */
```

```
33    /* vst_dispatch when final status is
         received. */
34    cmd = VS_Command_Create();
35    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
36    {
37       /* Send the command to the VolServ
         software. */
38       /* Note that status is not
         processed here. */
39       /* Instead, it is processed in the
         */
40       /* vst_dispatch routine. Also
         retry limit */
41       /* and priority are set as */
42       /* default parameters. */
43       rc = VSCMD_CreatePool(cmd,
44             VSID_DRIVEPOOL_NAME, dp,
45             VSID_DRIVE_ID_LIST, count,
         drivelist,
46             VSID_ENDFIELD);
47    }
48    return ( rc );
49 }
```


Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Create Pool request.

`VSCMD_CreatePool` does not trigger any MediaClass callbacks from VolServ.

The name specified for a new drive pool must be unique. A request to create a drive pool with a non-unique name will fail.

Drive pools can contain zero or more drives.

Drives belonging to a single drive pool can be associated with different archives. Drives are not required to be associated with an archive to belong to a drive pool.

Drive pools can contain drives that support incompatible media types.

If a drive pool is specified on a Mount request and the specified drive pool spans archives, VolServ may select a drive to honor the Mount request that is in a different archive than the medium that is selected to honor the request. If this occurs, a Move-Mount action is required. If permitted, the medium is scheduled for ejection from its parent archive and eventually entered into the archive associated with the assigned drive.

Whether or not Move-Mount action processing is permitted is specified at the archive level. The `ACTION_MODE` and `MOVEWAIT_OPTION` attributes control whether or not Move-Mount processing is allowed for a specific archive. These attributes are discussed under the `VS_Archive_SetFields` and `VS_Archive_GetFields` functions.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Create Pool request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

The `VSID_DRIVE_ID_LIST` and `VSID_COMP_STATE_LIST` parameters require that two arguments be passed instead of one.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Create Pool commands are set with `VSCMD_CreatePool_SetDefaults`. If command-specific defaults are set for Create Pool commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Create Pool command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Create Pool request:

- VSID_DRIVE_ID,
- VSID_DRIVE_ID_ENTRY,
- VSID_DRIVE_ID_TABLE,
- VSID_DRIVEPOOL_NAME,
- VSID_ERROR_CODE,

- VSID_ERROR_CODE_ENTRY,
- VSID_ERROR_CODE_TABLE,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l)`,
- `VS_Command_Create(l)`,
- `VS_Command_Destroy(l)`,
- `VS_Error_GetFields(l)`,
- `VS_Initialize(l)`,
- `VS_Status_GetFields(l)`,
- `VSCMD_DeletePool(l)`,
- `VSCMD_ModifyPool(l)`

VSCMD_ CreatePool_ SetDefaults

`VSCMD_CreatePool_SetDefaults` sets the command-level default parameters for Create Pool commands.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Create Pool commands are set with `VSCMD_CreatePool_SetDefaults`. If command-specific defaults are set for Create Pool commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Create Pool command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_CreatePool_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on Create Pool commands.
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	Name of the drive pool to be created. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_DRIVE_ID_LIST (int)	Number of drives to include in the new drive pool.
(VST_DRIVE_ID *)	Pointer to the list of drives to be included in the new drive pool.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Create Pool commands.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Create Pool commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Create Pool commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Create Pool commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Create Pool commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Create Pool commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_CreatePool_SetDefaults` returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_createpool_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_CreatePool API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 **************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_createpool_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_createpool_defaults()
17 #endif
18 {
19    VST_BOOLEAN            rc =
         VSE_FALSE;
```

```
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Create Drive Pool default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_CreatePool_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```


Notes

The `VSID_DRIVE_ID_LIST` and `VSID_COMP_STATE_LIST` parameters require that two arguments be passed instead of one.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l),`
- `VS_Error_GetFields(l),`
- `VS_Global_SetFields(l),`
- `VSCMD_CreatePool(l)`

VSCMD_ DeleteArchive MediaClass

`VSCMD_DeleteArchiveMediaClass` deletes an existing archive media class relationship.

Upon receipt of a `VSCMD_DeleteArchiveMediaClass` request, VolServ disassociates the archive media class relationship and returns status to the client indicating the success or failure of the request.

Synopsis

```
VST_BOOLEAN VSCMD_DeleteArchiveMediaClass
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)
```

Arguments

- handle = The command handle for the Delete Archive Media Class request.
- “…” = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive associated with the archive media class relationship to be deleted. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Name of the MediaClass group associated with the archive media class relationship to be deleted. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on this request.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_DeleteArchiveMediaClass returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.

  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code value is `VSE_VOLSERV`, the error occurred in VolServ and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API and the client uses `VST_ERROR_CODE` to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_deletearchivemediaclass_execu
       te
4  *
5  * PURPOSE:
6  * This executes the
       VSCMD_DeleteArchiveMediaClass
7  * API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
       vst_deletearchivemediaclass_execu
       te(void)
15 #else
16    VST_BOOLEAN
       vst_deletearchivemediaclass_execu
       te()
17 #endif
18 {
19    VST_BOOLEAN          rc = VSE_FALSE;
20    VST_ARCHIVE_NAME     archive;
21    VST_MEDIA_CLASS_NAME mediaclass;
22    VST_COMMAND_HANDLE   cmd;
23
24    /* get parameters from user */
25    printf("*** Delete Archive Media
       Class parameters ***\n" );
26    printf("Enter Archive Name ==> " );
27    gets( archive );
28    printf("Enter Media Class Name ==> "
       );
29    gets( mediaclass );
30    /* create the command handle */
```

```
31    /* Note that the command handle is
         not */
32    /* destoyed in this routine, but in
         */
33    /* vs_dispatch when final status is
         received. */
34    cmd = VS_Command_Create();
35    if (cmd != (VST_COMMAND_HANDLE )NULL)
36    {
37       /* Send the command to the VolServ
         software. */
38       /* Note that status is not
         processed here. */
39       /* Instead, it is processed in the
         */
40       /* vst_dispatch routine. Also,
         note that */
41       /* default values such as timeout,
         value */
42       /* limit and priority are
         setarchive */
43       /* retry mediaclassas default
         parameters. */
44       rc =
         VSCMD_DeleteArchiveMediaClass(cmd
         ,
45                VSID_ARCHIVE_NAME,
         archive,
46                VSID_MEDIA_CLASS_NAME,
         mediaclass,
47                VSID_ENDFIELD);
48    }
49    return ( rc );
50 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Delete Archive Media Class request.

`VSCMD_DeleteArchiveMediaClass` does not trigger any MediaClass callbacks from VolServ.

If the archive media class contains any media, the `VSCMD_DeleteArchiveMediaClass` request fails. This includes media that are currently marked for checkout or export and have not yet been ejected.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Delete Archive Media Class request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Delete Archive Media Class commands are set with `VSCMD_DeleteArchiveMediaClass_SetDefaults`. If

command-specific defaults are set for Delete Archive Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Delete Archive Media Class command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Delete Archive Media Class request:

- VSID_ARCHIVE_NAME,
- VSID_MEDIA_CLASS_NAME,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l)`,
- `VS_Command_Create(l)`,
- `VS_Command_Destroy(l)`,
- `VS_Error_GetFields(l)`,
- `VS_Initialize(l)`,
- `VS_Status_GetFields(l)`,
- `VSCMD_CreateArchiveMediaClass(l)`,
- `VSCMD_ModifyArchiveMediaClass(l)`

VSCMD_DeleteArchiveMediaClass_SetDefaults

`VSCMD_DeleteArchiveMediaClass_SetDefaults` sets the command-level default parameters for Delete Archive Media Class commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Delete Archive Media Class commands are set with `VSCMD_DeleteArchiveMediaClass_SetDefaults`. If command-specific defaults are set for Delete Archive Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Delete Archive Media Class command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_DeleteArchiveMedia
Class_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	Name of the archive associated with the archive media class relationship to be deleted. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on Delete Archive Media Class commands.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Name of the MediaClass group associated with the archive media class relationship to be deleted. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Delete Archive Media Class commands.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Delete Archive Media Class commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Delete Archive Media Class commands. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Delete Archive Media Class commands. Valid options are `VSE_TRUE` (API software waits for final status) and `VSE_FALSE` (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE`.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in `USER_FIELD` for Delete Archive Media Class commands. `USER_FIELD` is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Delete Archive Media Class commands. Neither the API software nor VolServ uses `USER_FIELD`.

Return Values

`VSCMD_DeleteArchiveMediaClass_SetDefaults` returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_deletearchivemediaclass_defau
       lts
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_DeleteArchiveMediaClass API
       call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
       vst_deletearchivemediaclass_defau
       lts(void)
15 #else
```

```
16    VST_BOOLEAN
         vst_deleteaarchivemediaclass_defa
         ults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Delete Archive Media
         Class default parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc =
         VSCMD_DeleteArchiveMediaClass_Set
         Defaults(
32          VSID_PRIORITY,
         priority,
33          VSID_USER_FIELD,
         user_field,
34          VSID_TIMEOUT_VALUE,
         timeout,
35          VSID_RETRY_LIMIT,
         retries,
36          VSID_STATUS_WAIT_FLAG,
         wait_flag,
37          VSID_ENTERPRISE_ID,
         enterprise_id,
38          VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l),`
- `VS_Error_GetFields(l),`
- `VS_Global_SetFields(l),`
- `VSCMD_DeleteArchiveMediaClass(l)`

VSCMD_ DeleteMedia Class

`VSCMD_DeleteMediaClass` deletes an existing MediaClass group from the VolServ system.

Upon receipt of a `VSCMD_DeleteMediaClass` command, VolServ removes the MediaClass group and returns status to the client indicating the success or failure of the request.

Synopsis

```
VST_BOOLEAN VSCMD_DeleteMediaClass
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)
```

Arguments

- handle = The command handle for the Delete Media Class request.
- “…” = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Name of the MediaClass group to be deleted. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on this request.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are `VSE_TRUE` (API software waits for final status) and `VSE_FALSE` (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE.`

Parameter Type	Description
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_DeleteMediaClass returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
    - If the object code value is `VSE_VOLSERV`, the error occurred in VolServ and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
    - If the object code value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API and the client uses `VST_ERROR_CODE` to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_deletemediaclass_execute
4  *
5  * PURPOSE:
6  * This executes the
       VSCMD_DeleteMediaClass API call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_deletemediaclass_execute(void
         )
14 #else
15    VST_BOOLEAN
         vst_deletemediaclass_execute()
16 #endif
17 {
18    VST_MEDIA_CLASS_NAME    mediaclass;
19    VST_COMMAND_HANDLE      cmd;
20    VST_BOOLEAN             rc =
         VSE_FALSE;
21
22    /* get parameters from user */
23     printf("*** Delete Media Class
         parameters ***\n" );
24    printf("\nEnter Media Class name to
         delete ==>");
25    gets( mediaclass);
26
27    /* create the command handle */
28    /* Note that the command handle is
         not */
29    /* destroyed in this routine, but in
         */
```

```
30    /* vst_dispatch when final status is
         received. */
31    cmd = VS_Command_Create();
32    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
33    {
34       /* Send the command to the VolServ
         software. */
35       /* Note that status is not
         processed here. */
36       /* Instead, it is processed in the
         */
37       /* vst_dispatch routine. Also,
         note that */
38       /* default values such as
         timeout,*/
39       /* value retry limit and priority
         are set as */
40       /* default parameters. */
41       rc = VSCMD_DeleteMediaClass(cmd,
42                VSID_MEDIA_CLASS_NAME,
         mediaclass,
43                VSID_ENDFIELD);
44    }
45    return ( rc );
46 }
```


Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Delete Media Class request.

`VSCMD_DeleteMediaClass` does not trigger any MediaClass callbacks from VolServ.

If the specified MediaClass group is associated with any archive media class relationship, the `VSCMD_DeleteMediaClass` request fails.

If the specified MediaClass group contains any media, the `VSCMD_DeleteMediaClass` request fails.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Delete Media Class request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Delete Media Class commands are set with `VSCMD_DeleteMediaClass_SetDefaults`. If command-specific defaults are set for Delete Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Delete Media Class command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Delete Media Class request:

- VSID_MEDIA_CLASS_NAME,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l),`
- `VS_Command_Create(l),`
- `VS_Command_Destroy(l),`
- `VS_Error_GetFields(l),`
- `VS_Initialize(l),`
- `VS_Status_GetFields(l),`
- `VSCMD_CreateMediaClass(l),`
- `VSCMD_ModifyMediaClass(l)`

VSCMD_DeleteMediaClass_SetDefaults

`VSCMD_DeleteMediaClass_SetDefaults` sets the command-level default parameters for Delete Archive Media Class commands.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and VS_Global_GetFields function calls.
- Command-specific parameter defaults for Delete Archive Media Class commands are set with VSCMD_DeleteMediaClass_SetDefaults. If command-specific defaults are set for Delete Archive Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Delete Media Class command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_DeleteMediaClass_
SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Rquired at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on Delete Archive Media Class commands.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	Name of the MediaClass group to delete. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Delete Archive Media Class commands.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Delete Archive Media Class commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Delete Archive Media Class commands. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Delete Archive Media Class commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Delete Archive Media Class commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Delete Archive Media Class commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_DeleteMediaClass_SetDefaults` returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_deletemediaclass_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_DeleteMediaClass API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_deletemediaclass_defaults(voi
         d)
15 #else
16    VST_BOOLEAN
         vst_deletemediaclass_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Create Archive Media
         Class default parameters ***\n" );
```

```
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc =
         VSCMD_DeleteMediaClass_SetDefault
         s(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l)`,
- `VS_Error_GetFields(l)`,
- `VS_Global_SetFields(l)`,
- `VSCMD_DeleteMediaClass(l)`

VSCMD_ DeletePool

`VSCMD_DeletePool` deletes an existing drive pool.

Upon receipt of a `VSCMD_DeletePool` request, VolServ removes the specified drive pool definition, as well as any drive pool member associations, and returns status to the client indicating the success or failure of the request.

Synopsis

```
VST_BOOLEAN VSCMD_DeletePool
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)
```

Arguments

- handle = The command handle for the Delete Pool request.
- “…” = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragrap*h*.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	Name of the drive pool to delete. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on this request.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are `VSE_TRUE` (API software waits for final status) and `VSE_FALSE` (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE.`

Parameter Type	Description
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_DeletePool returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
    - If the object code value is `VSE_VOLSERV`, the error occurred in VolServ and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
    - If the object code value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API and the client uses `VST_ERROR_CODE` to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_deletepool_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_DeletePool API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_deletepool_execute(void)
14 #else
15    VST_BOOLEAN vst_deletepool_execute()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_DRIVE_POOL_NAME     dp;
20    VST_COMMAND_HANDLE      cmd;
21
22    /* get parameters from user */
23    printf("*** Delete Pool Parameters
         ***\n" );
24    printf("\nEnter Drive Pool name to
         delete ==>");
25    gets( dp );
26
27    /* create the command handle */
28    /* Note that the command handle is
         not */
29    /* destoyed in this routine, but in
         */
30    /* vst_dispatch when final status is
         received. */
31    cmd = VS_Command_Create();
```

```
32    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
33    {
34       /* Send the command to the VolServ
         software. */
35       /* Note that status is not
         processed here. */
36       /* Instead, it is processed in the
         */
37       /* vst_dispatch routine. Also,
         note that */
38       /* default values such as
         timeout,*/
39       /* value retry limit and priority
         are set as */
40       /* default parameters. */
41       rc = VSCMD_DeletePool(cmd,
42
         VSID_DRIVEPOOL_NAME, dp,
43
         VSID_ENDFIELD);
44    }
45    return ( rc );
46 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Delete Pool request.

`VSCMD_DeletePool` does not trigger any MediaClass callbacks from VolServ.

Requests submitted prior to a Delete Pool request are not updated if a drive has already been allocated to the request.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Delete Pool request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Delete Pool commands are set with `VSCMD_DeletePool_SetDefaults`. If command-specific defaults are set for Delete Pool commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Delete Pool command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Delete Pool request:

- VSID_DRIVEPOOL_NAME,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l),`
- `VS_Command_Create(l),`
- `VS_Command_Destroy(l),`
- `VS_Error_GetFields(l),`
- `VS_Initialize(l),`
- `VS_Status_GetFields(l),`
- `VSCMD_CreatePool(l),`
- `VSCMD_ModifyPool(l)`

VSCMD_ DeletePool_ SetDefaults

`VSCMD_DeletePool_SetDefaults` sets the command-level default parameters for Delete Pool commands.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Delete Pool commands are set with `VSCMD_DeletePool_SetDefaults`. If command-specific defaults are set for Delete Pool commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Delete Pool command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_DeletePool_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on Delete Pool commands.
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	Name of the drive pool to delete. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Delete Pool commands.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Delete Pool commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Delete Pool commands. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Delete Pool commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Delete Pool commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Delete Pool commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_DeletePool_SetDefaults` returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_deletepool_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_DeletePool API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_deletepool_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_deletepool_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Delete Pool default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_DeletePool_SetDefaults(
```

```
32              VSID_PRIORITY,
          priority,
33              VSID_USER_FIELD,
          user_field,
34              VSID_TIMEOUT_VALUE,
          timeout,
35              VSID_RETRY_LIMIT,
          retries,
36              VSID_STATUS_WAIT_FLAG,
          wait_flag,
37              VSID_ENTERPRISE_ID,
          enterprise_id,
38              VSID_ENDFIELD);
39      return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l),`
- `VS_Error_GetFields(l),`
- `VS_Global_SetFields(l),`
- `VSCMD_DeletePool(l)`

VSCMD_ Disconnect

`VSCMD_Disconnect` disconnects a specified Internet address from the specified enterprise identifier. This association of Internet address with an enterprise identifier was made via the Connect command.

Even though the specified client Internet address is disassociated from the given enterprise identifier, the address can remain active for a different enterprise.

Upon receipt of a `VSCMD_Disconnect` request, VolServ verifies that the specified enterprise is associated with the given Internet address. If the association exists, VolServ removes the Internet address from the database and returns final status to inform the client that the request has been completed. If the association does not exist, the command fails, and failure status is returned (if possible) to the client.

Synopsis

```
VST_BOOLEAN VSCMD_Disconnect
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)
```

Arguments

- handle = The command handle for the Disconnect request.
- “…” = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_CONNECT_HANDLE (VST_CONNECT_HANDLE)	The connect handle that contains the enterprise callback address information of the enterprise whose connection to the VolServ system is broken. `VSID_CONNECT_HANDLE` is not applicable if `VSID_PROCEDURE_NUMBER`, `VSID_PROGRAM_NUMBER`, `VSID_PROTOCOL`, and `VSID_VERSION_NUMBER` are specified.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status for this request.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	The RPC procedure number of the client process to disconnect from VolServ. `VSID_PROCEDURE_NUMBER` is not applicable if `VSID_CONNECT_HANDLE` is specified.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	The RPC program number of the client process to disconnect from VolServ. `VSID_PROGRAM_NUMBER` is not applicable if `VSID_CONNECT_HANDLE` is specified.
VSID_PROTOCOL (VST_PROTOCOL)	The Internet protocol VolServ uses to return status messages and MediaClass callbacks to the client to be disconnected from VolServ. `VSID_PROTOCOL` is not applicable if `VSID_CONNECT_HANDLE` is specified.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_SOCKADDR_IN (VST_SOCKADDR_IN)	The Internet socket address for the client to disconnect from VolServ.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The enterprise identifier of the enterprise to disconnect from VolServ.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	A value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	The RPC version number of the client process to disconnect from VolServ.

Return Values

`VSCMD_Disconnect` returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed.A return code of VSE_FALSE (which is 0) means the command failed.
  - To determine where the error occurred, and what the error was, the client queries the command's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code value is `VSE_VOLSERV`, the error occurred in VolServ, and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API, and the client uses `VST_ERROR_CODE` to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.

- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_disconnect_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_Disconnect API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_disconnect_execute(void)
14 #else
15    VST_BOOLEAN vst_disconnect_execute()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_ENTERPRISE_ID
         TargetEnterpriseID;
20    VST_SOCKADDR_IN
         socketaddress;
```

```
21    VST_PROGRAM_NUMBER      prognum;
22    VST_COMMAND_HANDLE      cmd;
23    VST_VERSION_NUMBER      versnum;
24    VST_PROCEDURE_NUMBER    procnum;
25    int                     temp;
26
27    /* get parameters from user */
28    printf("*** Disconnect parameters
         ***\n" );
29    printf("Enter Enterprise ID ==> " );
30    TargetEnterpriseID =
         atoi(gets(input));
31    printf("Enter Program Number ==> " );
32    prognum = atoi(gets(input));
33    printf("Enter Version Number ==> " );
34    versnum = atoi(gets(input));
35    printf("Enter Procedure Number ==> "
         );
36    procnum = atoi(gets(input));
37    printf("Enter Socket sin family ==> "
         );
38    temp = atoi(gets(input));
39    socketaddress.sin_family = (short)
         temp;
40    printf("Enter Socket sin port ==> "
         );
41    temp = atoi(gets(input));
42    socketaddress.sin_port = (u_short)
         temp;
43    printf("Enter Socket sin address ==>
         " );
44    temp = atoi(gets(input));
45    socketaddress.sin_addr = (u_long)
         temp;
46
47    /* create the command handle */
48    /* Note that the command handle is
         not */
49    /* destoyed in this routine, but in
         */
50    /*  vst_dispatch when final status is
         received. */
```

```
51    cmd = VS_Command_Create();
52    if (cmd != (VST_COMMAND_HANDLE )NULL)
53    {
54       /* Send the command to the VolServ
          software. */
55       /* Note that status is not
          processed here. */
56       /* Instead, it is processed in the
          */
57       /* vst_dispatch routine. Also,
          note that */
58       /* default values such as
          timeout,*/
59       /* value retry limit and priority
          are set as */
60       /* default parameters. */
61       rc = VSCMD_Disconnect(cmd,
62          VSID_TARGET_ENTERPRISE_ID,
          TargetEnterpriseID,
63          VSID_PROGRAM_NUMBER, prognum,
64          VSID_VERSION_NUMBER, versnum,
65          VSID_PROCEDURE_NUMBER,
          procnum,
66          VSID_SOCKADDR_IN,
          socketaddress,
67          VSID_ENDFIELD);
68    }
69    return ( rc );
70 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Disconnect command.

The Disconnect command cannot trigger MediaClass callbacks from VolServ.

The VSID_CONNECT_HANDLE parameter may be used after a Connect Query command to disconnect an enterprise after the client has gone down.

A VSCMD_Disconnect request can be issued only through the client interface. The association between an enterprise and its clients cannot be established via the GUI.

A Disconnect request cannot be cancelled. The client may reestablish a connection by issuing a Connect request.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, final status for this command is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive final status on a Disconnect command submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.
- Command-specific parameter defaults for the Disconnect command are set with VSCMD_Disconnect_SetDefaults. If

command-specific defaults are set for the Disconnect command, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Disconnect command, the parameter identifier and the value to be used for the parameter can be submitted on the specific command itself.

The following fields can be retrieved from the status handle after a successful Disconnect request:

- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_TARGET_ENTERPRISE_ID,
- VSID_USER_FIELD.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l)`,
- `VS_Command_Create(l)`,
- `VS_Command_Destroy(l)`,
- `VS_Error_GetFields(l)`,
- `VS_Initialize(l)`,
- `VS_Status_GetFields(l)`,
- `VSCMD_Connect(l)`,
- `VSCMD_ConnectQuery(l)`

VSCMD_ Disconnect_ SetDefaults

`VSCMD_Disconnect_SetDefaults` sets the command-level default parameters for Disconnect commands.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Disconnect commands are set with `VSCMD_Disconnect_SetDefaults`. If command-specific defaults are set for Disconnect commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Disconnect command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_Disconnect_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on Disconnect commands.
VSID_CONNECT_HANDLE (VST_CONNECT_HANDLE)	The connect handle that contains the enterprise callback address information of the enterprise whose connection to the VolServ system is broken. `VSID_CONNECT_HANDLE` is not applicable if `VSID_PROCEDURE_NUMBER`, `VSID_PROGRAM_NUMBER`, `VSID_PROTOCOL`, and `VSID_VERSION_NUMBER` are specified.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Disconnect commands.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Disconnect commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	The RPC procedure number of the client process to disconnect from VolServ. `VSID_PROCEDURE_NUMBER` is not applicable if `VSID_CONNECT_HANDLE` is specified.

Parameter Type	Description
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	The RPC program number of the client process to disconnect from VolServ. VSID_PROGRAM_NUMBER is not applicable if VSID_CONNECT_HANDLE is specified.
VSID_PROTOCOL (VST_PROTOCOL)	The Internet protocol VolServ uses to return status messages and MediaClass callbacks to the client to be disconnected from VolServ. VSID_PROTOCOL is not applicable if VSID_CONNECT_HANDLE is specified.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Disconnect commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_SOCKADDR_IN (VST_SOCKADDR_IN)	The Internet socket address for the client to disconnect from VolServ.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	The Internet socket address for the client to disconnect from VolServ. Flag indicating whether the API software waits for final status from VolServ (or times-out) for Disconnect commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The enterprise identifier of the enterprise to disconnect from VolServ.

Parameter Type	Description
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in `USER_FIELD` for Disconnect commands. `USER_FIELD` is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Disconnect commands. Neither the API software nor VolServ uses `USER_FIELD`.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	The RPC version number of the client process to disconnect from VolServ.

Return Values

`VSCMD_DeletePool_SetDefaults` returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_disconnect_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Disconnect API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_disconnect_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_disconnect_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Disconnect default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_Disconnect_SetDefaults(
```

```
32              VSID_PRIORITY,
          priority,
33              VSID_USER_FIELD,
          user_field,
34              VSID_TIMEOUT_VALUE,
          timeout,
35              VSID_RETRY_LIMIT,
          retries,
36              VSID_STATUS_WAIT_FLAG,
          wait_flag,
37              VSID_ENTERPRISE_ID,
          enterprise_id,
38              VSID_ENDFIELD);
39      return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l),`
- `VS_Error_GetFields(l),`
- `VS_Global_SetFields(l),`
- `VSCMD_Connect(l),`
- `VSCMD_ConnectQuery(l)`

VSCMD_ Dismount

A `VSCMD_Dismount` request informs VolServ that the client is finished using a drive and the medium mounted in the drive.

Upon receipt of a `VSCMD_Dismount` request for an automated archive, VolServ determines whether the medium has been ejected from the drive by the storage subsystem.

If the medium has been ejected from the drive, VolServ commands the archive robotics to move the medium from the drive pickup point to a bin within the archive system. A successful return code is returned to the client after the medium movement has completed.

If the medium has not been ejected from the drive, the dismount request fails, and VolServ returns a failure status to the client.

For manual archives, a dismount notice is sent to the appropriate archive's console display for action. An operator must dismount the specified medium and then notify VolServ the medium dismount is complete. VolServ returns status to the client only after the operator confirms the dismount is complete.

The Dismount command supports a lock identifier parameter. This parameter is required if the drive to be dismounted has been previously locked with a Lock request.

Synopsis

```
VST_BOOLEAN VSCMD_Dismount
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)
```

Arguments

- handle = The command handle for the Dismount request.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_DRIVE_ID (VST_DRIVE_ID)	Identifier of the drive to dismount.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on this request.
VSID_ERROR_COUNT (VST_COUNT)	Number of read/write errors encountered while the drive was mounted.
VSID_LOCK_ID (VST_LOCK_ID)	The drive's lock identifier, required if a drive is locked.
VSID_MEDIA_ID (VST_MEDIA_ID)	Identifier of the medium to dismount.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.
VSID_USAGE_TIME (VST_USAGE)	The length of time, in seconds, the drive was in use.

Return Values

`VSCMD_Dismount` returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code value is `VSE_VOLSERV`, the error occurred in VolServ and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API and the client uses `VST_ERROR_CODE` to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.

- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_dismount_execute
4  *
5  * PURPOSE:
6  * This routine tests the VSCMD_Dismount
       API call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_dismount_execute(
         void )
14 #else
15    VST_BOOLEAN vst_dismount_execute()
16 #endif
17 {
18    VST_BOOLEAN            rc =
         VSE_FALSE;
19    VST_MEDIA_ID           media;
20    VST_DRIVE_ID           drive;
21    VST_LOCK_ID            lock;
22    VST_USAGE              time;
```

```
23    VST_COUNT               err;
24    VST_COMMAND_HANDLE      cmd;
25
26    /* get parameters from user */
27    printf("*** Dismount parameters
         ***\n" );
28    printf("Enter Media ID ==> " );
29    gets( media );
30    printf("Enter Drive ID ==> " );
31    drive = atoi(gets(input));
32    printf("Enter Lock ID ==> " );
33    lock = atol(gets(input));
34    printf("Enter Usage Time ==> " );
35    time = atol(gets(input));
36    printf("Enter Error Count ==> " );
37    err = atoi(gets(input));
38    /* create the command handle */
39    /* Note that the command handle is
         not */
40    /* destoyed in this routine, but in
         */
41    /* vst_dispatch when final status is
         received. */
42    cmd = VS_Command_Create();
43    if (cmd!= (VST_COMMAND_HANDLE )NULL)
44    {
45       /* Send the command to the VolServ
         software. */
46       /* Note that status is not
         processed here. */
47       /* Instead, it is processed in the
         */
48       /* vst_dispatch routine. Also,
         note that */
49       /* default values such as
         timeout,*/
50       /* value retry limit and priority
         are set as */
51       /* default parameters. */
52       rc = VSCMD_Dismount(cmd,
53                           VSID_DRIVE_ID,
         drive,
```

```
54                          VSID_MEDIA_ID,
        media,
55                          VSID_LOCK_ID,
        lock,
56
        VSID_USAGE_TIME, time,
57
        VSID_ERROR_COUNT, err,
58                          VSID_ENDFIELD);
59    }
60    return ( rc );
61 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Dismount request.

`VSCMD_Dismount` can trigger MediaClass callbacks from VolServ.

`VSID_USAGE_TIME` and `VSID_ERROR_COUNT` are optional parameters provided for the client that wants to maintain drive statistics. Correct values for these parameters are the responsibility of the client. When VolServ detects these parameters on a Dismount command, the usage time and/or error count fields for the specified drive are incremented by the amount specified on the Dismount command.

A Dismount request cannot be cancelled. If necessary, the client may request the medium be remounted by issuing a Mount request.

If the drive specified in a Dismount request is off-line, unavailable, or in the diagnostic state, the Dismount request fails.

If the lock identifier specified on a Dismount request differs from the lock identifier associated with the specified drive, the Dismount request fails.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Dismount request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Dismount commands are set with `VSCMD_Dismount_SetDefaults`. If command-specific defaults are set for Dismount commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Dismount command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Dismount request:

- VSID_DRIVE_ID,
- VSID_DRIVE_ID_ENTRY,
- VSID_DRIVE_ID_TABLE,
- VSID_LOCK_ID,
- VSID_MEDIA_ID,
- VSID_MEDIA_ID_ENTRY,
- VSID_MEDIA_ID_TABLE,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l),`
- `VS_Command_Create(l),`
- `VS_Command_Destroy(l),`
- `VS_Error_GetFields(l),`
- `VS_Initialize(l),`
- `VS_Status_GetFields(l),`
- `VSCMD_Dismount_SetDefaults(l),`
- `VSCMD_Mount(l),`
- `VSCMD_Lock(l)`

VSCMD_ Dismount_Set Defaults

`VSCMD_Dismount_SetDefaults` sets the command-level default parameters for Dismount commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Dismount commands are set with `VSCMD_Dismount_SetDefaults`. If command-specific defaults are set for Dismount commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Dismount command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_Dismount_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on Dismount commands.
VSID_DRIVE_ID (VST_DRIVE_ID)	Identifier of the drive to dismount.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Dismount commands.
VSID_ERROR_COUNT (VST_COUNT)	Number of read/write errors encountered while the drive was mounted.
VSID_LOCK_ID (VST_LOCK_ID)	The drive's lock identifier, required if a drive is locked.
VSID_MEDIA_ID (VST_MEDIA_ID)	Identifier of the medium to dismount.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on this request.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Dismount commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Dismount commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Dismount commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Dismount commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Dismount commands. Neither the API software nor VolServ uses USER_FIELD.
VSID_USAGE_TIME (VST_USAGE)	The length of time, in seconds, the drive was in use.

Return Values

`VSCMD_Dismount_SetDefaults` returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_dismount_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Dismount API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 **************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_dismount_defaults(void)
15 #else
16    VST_BOOLEAN vst_dismount_defaults()
17 #endif
18 {
19    VST_BOOLEAN            rc =
         VSE_FALSE;
20    VST_PRIORITY           priority;
```

```
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Dismount default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_Dismount_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39  return ( rc );
40 }
```


Notes

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- `vsapi(l)`,
- `VS_Error_GetFields(l)`,
- `VS_Global_SetFields(l)`,
- `VSCMD_Dismount(l)`

VSCMD_DrivePoolQuery

The `VSCMD_DrivePoolQuery` queries for information about one drive pool or about all drive pools known to the VolServ system.

Upon receipt of a Drive Pool Query request, VolServ obtains the requested information about the specified drive pool and returns this information to the client.

Synopsis

```
VST_BOOLEAN VSCMD_DrivePoolQuery
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)
```


Arguments

- handle = The command handle for the Drive Pool Query request.
- “…” = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_DRIVEPOOL_NAME (VST_DRIVEPOOL_NAME)	Name of the drive pool to query. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on this request.
VSID_POOL_QUERY_OPT (VST_QUERY_LIST_OPTION)	Specifies what drive information, if any, is requested for each specified drive pool. The client can request no drive information, a list of drive identifiers associated with each drive pool, or detailed information about each drive associated with each drive pool. Valid `VSID_POOL_QUERY_OPT` values are enumerated in the *vs_types.h* file.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is being requested for a single specified drive pool or for all drive pools. Valid `VSE_QUERY_OPTION` values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_DrivePoolQuery returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.

  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code value is `VSE_VOLSERV`, the error occurred in VolServ and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API and the client uses `VST_ERROR_CODE` to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_drivepoolquery_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_DrivePoolQuery
       API call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_drivepoolquery_execute(void)
14 #else
15    VST_BOOLEAN
         vst_drivepoolquery_execute()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_QUERY_OPTION        queryopt;
20    VST_QUERY_LIST_OPTION
         querylistopt;
21    int                     count;
22    VST_DRIVE_POOL_NAME     drivepool;
23    VST_COMMAND_HANDLE      cmd;
24
25    /* get parameters from user */
26    printf("*** DrivePool Query
         parameters ***\n" );
27    printf("0) Query by drive pool Name,
         1) Query all ==> " );
28    queryopt = atoi(gets(input));
29
30    if (queryopt == 0)
31    {
32       printf("\nEnter drive pool Name
         ==>");
```

```
33          gets( drivepool);
34       }
35       printf(“\n1) DriveList , 2) DriveList
            Details ==> “ );
36       querylistopt = atoi(gets(input));
37
38       /* create the command handle */
39       /* Note that the command handle is
            not */
40       /* destroyed in this routine, but in
            */
41       /* vst_dispatch when final status is
            received. */
42       cmd = VS_Command_Create();
43       if ( cmd != (VST_COMMAND_HANDLE)
            NULL)
44       {
45          /* Send the command to the VolServ
            software. */
46          /* Note that status is not
            processed here. */
47          /* Instead, it is processed in the
            */
48          /* vst_dispatch routine. Also,
            note that */
49          /* default values such as
            timeout,*/
50          /* value retry limit and priority
            are set as */
51          /* default parameters. */
52          if (queryopt == 0)
53          {
54             /* query a single drive pool */
55             rc = VSCMD_DrivePoolQuery(cmd,
56                VSID_QRY_OPTION,
            queryopt,
57                VSID_POOL_QRY_OPT,
            querylistopt,
58                VSID_DRIVEPOOL_NAME,
            drivepool,
59                VSID_ENDFIELD);
60          }
```

```
61       else
62       {
63          /* query all drive pools */
64             rc =
         VSCMD_DrivePoolQuery(cmd,
65                VSID_QRY_OPTION,
         queryopt,
66                VSID_POOL_QRY_OPT,
         querylistopt,
67                VSID_ENDFIELD);
68       }
69    }
70    return ( rc );
71 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generate intermediate status in response to a Drive Pool Query request. Statuses are cumulative. Each status is appended to the previous status so that after the final status is given, the status handle contains information from all statuses.

`VSCMD_CreatePool` does not trigger any MediaClass callbacks from VolServ.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Drive Pool Query request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Drive Pool Query commands are set with `VSCMD_DrivePoolQuery_SetDefaults`. If command-specific defaults are set for Drive Pool Query commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Drive Pool Query command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Drive Pool Query request:

- VSID_DRIVEPOOL_HANDLE,
- VSID_DRIVEPOOL_HANDLE_ENTRY,
- VSID_DRIVEPOOL_HANDLE_TABLE,
- VSID_QUERY_OPTION,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l)`,
- `VS_Command_Create(l)`,
- `VS_Command_Destroy(l)`,
- `VS_Command_GetFields(l)`,
- `VS_Error_GetFields(l)`,
- `VS_Initialize(l)`,
- `VS_Status_GetFields(l)`,
- `VS_Table_GetFields(l)`,
- `VSCMD_DrivePoolQuery_SetDefaults(l)`

VSCMD_DrivePoolQuery_SetDefaults

`VSCMD_DrivePoolQuery_SetDefaults` sets the command-level default parameters for Drive Pool Query commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Drive Pool Query commands are set with `VSCMD_DrivePoolQuery_SetDefaults`. If command-specific defaults are set for Drive Pool Query commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Drive Pool Query command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN
VSCMD_DrivePoolQuery_SetDefaults (
“…”,
VSID_ENDFIELD)
```

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on Drive Pool Query commands.
VSID_DRIVEPOOL_NAME (VST_DRIVEPOOL_NAME)	Name of the drive pool to query. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Drive Pool Query commands.
VSID_POOL_QUERY_OPT (VST_QUERY_LIST_OPTION)	Specifies what drive information, if any, is requested for each specified drive pool. The client can request no drive information, a list of drive identifiers associated with each drive pool, or detailed information about each drive associated with each drive pool. Valid `VSID_POOL_QUERY_OPT` values are enumerated in the *vs_types.h* file.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Drive Pool Query commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is being requested for a single specified drive pool or for all drive pools. Valid VSE_QUERY_OPTION values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Drive Pool Query commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Drive Pool Query commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to be put in USER_FIELD for Drive Pool Query commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Drive Pool Query commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_DrivePoolQuery_SetDefaults` returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_drivepoolquery_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_DrivePoolQuery API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 **************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_drivepoolquery_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_drivepoolquery_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
```

```
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** drive pool Query default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc =
         VSCMD_DrivePoolQuery_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l)`,
- `VS_Error_GetFields(l)`,
- `VS_Global_SetFields(l)`,
- `VSCMD_DrivePoolQuery(l)`

VSCMD_Drive Query

`VSCMD_DriveQuery` queries for information about one or more drives known to the VolServ system.

Upon receipt of a Drive Query command, VolServ obtains the requested information and returns this information to the client.

Synopsis

```
VST_BOOLEAN VSCMD_DriveQuery
(VST_COMMAND_HANDLE handle,
“…”,
VSID_ENDFIELD)
```

Arguments

- handle = The command handle for the Drive Query request.
- “…” = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status for this request.
VSID_DRIVE_ID (VST_DRIVE_ID)	Identifier of a single drive to query.
VSID_DRIVE_ID_LIST (int)	Number of drives to query. Can be greater than or equal to 1.
(VST_DRIVE_ID *)	Pointer to a list of identifiers of drives to query.

Parameter Type	Description
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive final status on this request.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is requested for all drives known to the VolServ system or for the drives identified in a list specified with the Drive Query request. Valid `VSE_QUERY_OPTION` values are enumerated in the *vs_types.h* file.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are `VSE_TRUE` (API software waits for final status) and `VSE_FALSE` (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE.`
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	Value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_DriveQuery returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code value is VSE_VOLSERV, the error occurred in VolServ and the client uses VST_ERROR_NUMCODE to identify the specific error.

  - If the object code value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API and the client uses VST_ERROR_CODE to identify the specific error.

- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_drivequery_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_DriveQuery API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_drivequery_execute(void)
14 #else
15    VST_BOOLEAN vst_drivequery_execute()
```

```
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_QUERY_OPTION     queryopt;
20    int                  count;
21    VST_DRIVE_ID         temp_drive_id;
22    VST_DRIVE_ID
         drivelist[VST_MAX_ITEMS];
23    VST_COMMAND_HANDLE   cmd;
24
25    /* get parameters from user */
26    printf("*** Drive Query parameters
         ***\n" );
27    printf("0) Query by drive list, 1)
         Query all, 2) Query single
         DriveID==> " );
28    queryopt = atoi(gets(input));
29
30    if (queryopt == 0)
31    {
32       count =
         vst_getdrivelist(drivelist,
         VST_MAX_ITEMS);
33    }
34    else if (queryopt == 2)
35    {
36       printf("\nEnter Drive ID to query:
         ");
37       temp_drive_id =
         atoi(gets(input));
38    }
39
40    /* create the command handle */
41    /* Note that the command handle is
         not */
42    /* destroyed in this routine, but in
         */
43    /* vst_dispatch when final status is
         received. */
44    cmd = VS_Command_Create();
45    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
```

```
46    {
47       /* Send the command to the VolServ
         software. */
48       /* Note that status is not
         processed here. */
49       /* Instead, it is processed in the
         */
50       /* vst_dispatch routine Also, note
         that */
51       /* default values such as
         timeout,*/
52       /* value retry limit and priority
         are set as */
53       /* default parameters. */
54       if(queryopt == 2)
55       {
56          rc = VSCMD_DriveQuery(cmd,
57             VSID_QRY_OPTION,
         VSE_QUERY_OPTION_NONE,
58             VSID_DRIVE_ID,
         temp_drive_id,
59             VSID_ENDFIELD);
60       }
61       else
62       {
63          rc = VSCMD_DriveQuery(cmd,
64             VSID_QRY_OPTION,
         queryopt,
65             VSID_DRIVE_ID_LIST,
         count,drivelist,
66             VSID_ENDFIELD);
67       }
68    }
69    return ( rc );
70 }
```


Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generate intermediate status in response to a Drive Query request. Statuses are cumulative. Each status is added to the previous status so that after the final status is given, the status handle contains information from all statuses.

`VSCMD_DriveQuery` does not trigger any MediaClass callbacks from VolServ.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

The `VSID_DRIVE_ID_LIST` and `VSID_COMP_STATE_LIST` parameters require that two arguments be passed instead of one.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Drive Query request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Drive Query commands are set with `VSCMD_DriveQuery_SetDefaults`. If

command-specific defaults are set for Drive Query commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Drive Query command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Drive Query request:

- VSID_DRIVE_HANDLE,
- VSID_DRIVE_HANDLE_ENTRY,
- VSID_DRIVE_HANDLE_TABLE,
- VSID_ERROR_CODE,
- VSID_ERROR_CODE_ENTRY,
- VSID_ERROR_CODE_TABLE,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l)`,
- `VS_Command_Create(l)`,
- `VS_Command_Destroy(l)`,
- `VS_Command_GetFields(l)`,
- `VS_Drive_GetFields(l)`,
- `VS_Error_GetFields(l)`,
- `VS_Initialize(l)`,
- `VS_Status_GetFields(l)`,
- `VS_Table_GetFields(l)`,
- `VSCMD_DriveQuery_SetDefaults(l)`

VSCMD_DriveQuery_SetDefaults

`VSCMD_DriveQuery_SetDefaults` sets the command-level default parameters for Drive Query commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Drive Query commands are set with `VSCMD_DriveQuery_SetDefaults`. If command-specific defaults are set for Drive Query commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Drive Query command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_DriveQuery_SetDefaults
(
“…”,
VSID_ENDFIELD)
```

Arguments

- "…" = Variable length argument list consisting of pairs of arguments. Each pair of arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	Name of the client dispatch routine to receive status on Drive Query commands.
VSID_DRIVE_ID (VST_DRIVE_ID)	Identifier of a single drive to query.
VSID_DRIVE_ID_LIST (int)	Number of drives to query. May be greater than or equal to 1.
(VST_DRIVE_ID *)	Pointer to a list of identifiers of drives to query.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	Identifier of the enterprise, if any, to receive intermediate and final status on Drive Query commands.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Drive Query commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_QUERY_OPTION (VST_QUERY_OPTION)	Indicates whether information is being requested for all drives known to the VolServ system or for the drives identified in a list specified with the Drive Query request. Valid `VSE_QUERY_OPTION` values are enumerated in the *vs_types.h* file.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	Number of times the API software retries for command status from VolServ before returning a time-out to the client software for Drive Query commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	Flag indicating whether the API software waits for final status from VolServ (or times-out) for Drive Query commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	Amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	Value to put in USER_FIELD for Drive Query commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Drive Query commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_DriveQuery_SetDefaults` returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - Value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_drivequery_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_DriveQuery API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 **************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_drivequery_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_drivequery_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
```

```
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Drive Query default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_DriveQuery_SetDefaults(
32           VSID_PRIORITY,
         priority,
33           VSID_USER_FIELD,
         user_field,
34           VSID_TIMEOUT_VALUE,
         timeout,
35           VSID_RETRY_LIMIT,
         retries,
36           VSID_STATUS_WAIT_FLAG,
         wait_flag,
37           VSID_ENTERPRISE_ID,
         enterprise_id,
38           VSID_ENDFIELD);
39    return ( rc );
40 }
```


Notes

The `VSID_DRIVE_ID_LIST` and `VSID_COMP_STATE_LIST` parameters require that two arguments be passed instead of one.

Two levels of default parameter settings are used in the API software— global defaults and command-specific defaults.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- `vsapi(l),`
- `VS_Error_GetFields(l),`
- `VS_Global_SetFields(l),`
- `VSCMD_DriveQuery(l)`

VSCMD_ Modify ArchiveMedia Class

`VSCMD_ModifyArchiveMediaClass` modifies the value of one or more attributes of an archive media class. An archive media class is the association of a MediaClass group with a defined archive.

All archive media class attributes must be specified on a Modify Archive Media Class request, whether the value of an attribute is being modified or not.

Synopsis

VST_BOOLEAN VSCMD_ModifyArchiveMediaClass
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The command handle for this Modify Archive Media Class command.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_ACTION (VST_ARCHIVE_ACTION_MODE)	Specifies what action VolServ is to take when the number of media in the archive media class exceeds the specified high mark threshold. Valid `VSID_ARCHIVE_ACTION` values are enumerated in the *vs_types.h* file.
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The name of the archive associated with the archive media class. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CAPACITY (VST_CAPACITY)	The percentage of the total MediaClass capacity that can be stored in this archive.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.
VSID_COMPONENT_HANDLE_TABLE (VST_TABLE_HANDLE)	Preferred locations (in table format) for media assigned to this archive media class
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status for this request.
VSID_HIGH_MARK (VST_HIGH_MARK)	The percentage of `VSID_CAPACITY` above which the specified migration policy option is performed or initiated. This field is applicable only if `VSID_ARCHIVE_ACTION` is set to `VSE_ARCHIVE_ACTION_NOTIFY` or `VSE_ARCHIVE_ACTION_MIG`.
VSID_LOW_MARK (VST_LOW_MARK)	The percentage of `VSID_CAPACITY` below which the specified migration policy option is performed or terminated. This field is applicable only if `VSID_ARCHIVE_ACTION` is set to `VSE_ARCHIVE_ACTION_NOTIFY` or `VSE_ARCHIVE_ACTION_MIG`.

Parameter Type	Description
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The name of the MediaClass group associated with the archive media class. Valid MediaClass names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_MIGRATION_PRIORITY (VST_PRIORITY)	The migration priority applied to this archive media class.
VSID_PERCENT (VST_PERCENT)	The percentage of the MediaClass capacity allowed in the archive associated with the archive media class.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (times-out) for this request. Valid options are `VSE_TRUE` (API software waits for final status) and `VSE_FALSE` (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE.`

Parameter Type	Description
VSID_TARGET_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The destination archive for media automatically migrated out of this archive media class. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	A value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_ModifyArchiveMediaClass returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed.
  A return code of VSE_FALSE (which is 0) means the command failed.
  To determine where the error occurred, and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.

  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
        *********
2  *
```

```
3  * FUNCTION:
         vst_modarchivemediaclass_execute
4  *
5  * PURPOSE:
6  * This executes the
         VSCMD_ModifyArchiveMediaClass
7  * API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 ************************************
         *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_modarchivemediaclass_execute(
         void)
15 #else
16    VST_BOOLEAN
         vst_modarchivemediaclass_execute(
         )
17 #endif
18 {
19    int                          i;
20    int                          count;
21    VST_BOOLEAN                  rc =
         VSE_FALSE;
22    VST_ARCHIVE_NAME             archive;
23    VST_MEDIA_CLASS_NAME
         mediaclass;
24    VST_CAPACITY                 capacity;
25    VST_ARCHIVE_ACTION_OPTION    action;
26    VST_HIGH_MARK                highmark;
27    VST_LOW_MARK                 lowmark;
28    VST_PRIORITY                 migpri;
29    VST_ARCHIVE_NAME
         targetarchive;
30    VST_TABLE_HANDLE
         comphandletable;
31    VST_COMPONENT_HANDLE
         comphandle;
```

```
32    VST_COMP_TYPE  CompType =
         VSE_COMPTYPE_COLUMN;
33    VST_COMPONENT_ID           CompID;
34    VST_COMMAND_HANDLE         cmd;
35
36    bzero ( CompID, sizeof (
         VST_COMPONENT_ID ) );
37    /* get parameters from user */
38    printf("*** Modify parameters ***\n"
         );
39    printf("*** The archive media class
         grouping must exist. ***\n");
40    printf("Enter Archive Name ==> " );
41    gets( archive );
42    printf("Enter Media Class Name ==> "
         );
43    gets( mediaclass );
44    printf("Enter Capacity Percent ==> "
         );
45    capacity = atoi(gets(input));
46    printf("Enter Archive action option
         (0-none/1-mig/2-notify) ==> " );
47    action = atoi(gets(input));
48    printf("Enter High Mark Percentage
         ==> " );
49    highmark = atoi(gets(input));
50    printf("Enter Low Mark Percentage ==>
         " );
51    lowmark = atoi(gets(input));
52
53    if ( action == VSE_ARCHIVE_ACTION_MIG
         )
54    {
55       printf("Enter Target Archive ==> "
         );
56       gets( targetarchive );
57       printf("Enter Migration Priority
         == > " );
58       migpri = atoi(gets(input));
59       /* These only need to be set when
         migration */
60       /* is used. */
```

```
61
       VSCMD_ModifyArchiveMediaClass_Set
       Defaults (
62        VSID_TARGET_ARCHIVE_NAME,
       targetarchive,
63        VSID_MIGRATION_PRIORITY,
       migpri,
64        VSID_ENDFIELD );
65    }
66
67    printf("How many prefered placements
       (0 to skip): ");
68    count = atoi(gets(input));
69    if (count > 0)
70    {
71       comphandletable =
       VS_Table_Create(VSE_COMPONENT_HAN
       DLE, count);
72       if (comphandletable ==
       (VST_TABLE_HANDLE) NULL)
73       {
74          return (VSE_FALSE);
75       }
76       for (i = 0; i < count; i++)
77       {
78          printf("Enter row #%d:", i +
       1);
79          CompID[0] = (short)
       atoi(gets(input));
80          printf("Enter column #%d:", i +
       1);
81          CompID[1] = (short)
       atoi(gets(input));
82          CompID[2] = 0;
83          CompID[3] = 0;
84          comphandle =
       VS_Component_Create();
85
       VS_Component_SetFields(comphandle
       ,
86                         VSID_COMP_TYPE,
             CompType,
```

```
87                          VSID_COMP_ID,
      CompID,
88                          VSID_ENDFIELD);
89
      VS_Table_AddEntry(comphandletable
      ,comphandle);
90    }
91
      VSCMD_ModifyArchiveMediaClass_Set
      Defaults(
92
      VSID_COMPONENT_HANDLE_TABLE,
      comphandletable,
93          VSID_ENDFIELD);
94  }
95
96  /* create the command handle */
97  /* Note that the command handle is
       not */
98  /* destoyed in this routine, but in
       */
99  /* vst_dispatch when final status is
       received. */
100 cmd = VS_Command_Create();
101 if (cmd != (VST_COMMAND_HANDLE )NULL)
102 {
103    /* Send the command to the VolServ
       software. */
104    /* Note that status is not
       processed here. */
105    /* Instead, it is processed in the
       */
106    /* vst_dispatch routine. Also,
       note that */
107    /* default values such as timeout
       */
108    /* value retry limit and priority
       are set as */
109    /* default parameters. */
110    rc =
       VSCMD_ModifyArchiveMediaClass(cmd
       ,
```

```
111                 VSID_ARCHIVE_NAME,
          archive,
112                 VSID_MEDIA_CLASS_NAME,
          mediaclass,
113                 VSID_HIGH_MARK,
          highmark,
114                 VSID_LOW_MARK,
          lowmark,
115                 VSID_CAPACITY,
          capacity,
116                 VSID_ENDFIELD);
117    }
118    return ( rc );
119}
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Modify Archive Media Class.

`VSCMD_ModifyArchiveMediaClass` does not trigger unsolicited MediaClass callbacks from VolServ.

The migration policy options for are no action, operator notification, and automatic migration.

When the number of media in an archive media class reaches the high mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy option is set to notify.
- Initiates automatic migration of media if the migration policy is set to migrate.

When the number of media in an archive media class drops to the low mark threshold, VolServ:

- Does nothing if the migration policy option is set to none.
- Issues an operator message if the migration policy is set to notify.
- Terminates automatic migration of media if the migration policy is set to migrate.

The capacity value of an archive media class is relative to the MediaClass group specified overall capacity. Consideration should be given to all media classes that are able to share this archive to ensure that reasonably comparable capacity limitations and high/low marks are set for each archive media class.

Media can reside in an archive only if their associated MediaClass group has an archive media class assignment in that archive.

Archive media class computed capacity limits are “soft”, that is, they can be exceeded when media are imported or moved in from another archive. If automigration is specified, media of this MediaClass group is marked for movement to their target archive. The media type capacity designates the “hard” limit when entering media into an archive.

An archive media class computed capacity is refigured if the capacity of a MediaClass group changes.

Checks to determine if the high mark has been reached or exceeded or the low mark has been reached or passed occur:

- After any Eject, Enter, Reclassify, or Modify Archive Media Class command executes.
- After the MediaClass group or archive media class are redefined.

The sum of all archive media class capacities can exceed the archive's physical ability to house media. If VSID_CAPACITY values are set unrealistically high and VSID_HIGH_MARK is similarly high, the archive may physically completely fill before any automigration procedure is triggered.

Components listed as preferred for storage of media of this MediaClass group do not have exclusive ownership of those components.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive final status on a Modify Archive Media Class request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.

Note

Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.

- Command-specific parameter defaults for Modify Media Class commands are set with `VSCMD_ModifyMediaClass_SetDefaults`. If command-specific defaults are set for Modify Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Modify Media Class command, the parameter identifier and the value used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Modify Archive Media Class request:

- `VSID_ARCHIVE_NAME`,
- `VSID_COMPONENT_HANDLE`,
- VSID_COMPONENT_HANDLE_ENTRY,
- VSID_COMPONENT_HANDLE_TABLE,
- `VSID_ERROR_CODE`,
- `VSID_ERROR_CODE_ENTRY`,
- VSID_ERROR_CODE_TABLE,
- `VSID_MEDIA_CLASS_NAME`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_CreateArchiveMediaClass(l),
- VSCMD_DeleteArchiveMediaClass(l),
- VSCMD_ModifyArchiveMediaClass_SetDefaults(l)

VSCMD_ Modify ArchiveMedia Class_Set Defaults

`VSCMD_ModifyArchieMediaClass_SetDefaults` sets the command-level default parameters for Modify Archive Media Class commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Modify Media Class commands are set with `VSCMD_ModifyMediaClass_SetDefaults`. If command-specific defaults are set for Modify Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Modify Media Class command, the parameter identifier and the value used for the parameter can be submitted on the specific request itself

Synopsis

VST_BOOLEAN VSCMD_ModifyArchiveMediaClass
_SetDefaults
(“…”,
VSID_ENDFIELD)

Arguments

- **“…”** = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value used as a command default

value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_ARCHIVE_ACTION (VST_ARCHIVE_ACTION_MODE)	Specifies what action VolServ is to take when the number of media in the archive media class exceeds the specified high mark threshold. Valid VSID_ARCHIVE_ACTION values are enumerated in the *vs_types.h* file.
VSID_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The name of the archive associated with the archive media class. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_CAPACITY (VST_CAPACITY)	The percentage of the total MediaClass capacity that can be stored in this archive.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on Modify Archive Media Class commands.
VSID_COMPONENT_HANDLE_TABLE (VST_TABLE_HANDLE)	Preferred locations (in table format) for media assigned to this archive media class.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on Modify Archive Media Class commands.

Parameter Type	Description
VSID_PRIORITY)	The requested execution priority for Modify Archive Media Class commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Modify Archive Media Class commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Modify Archive Media Class commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The destination archive for media automatically migrated out of this archive media class. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for Modify Archive Media Class commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Modify Archive Media Class commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_ModifyArchiveMediaClass_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION:
       vst_modarchivemediaclass_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
```

```
7  * VSCMD_ModifyArchiveMediaClass API
       call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 ************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
       vst_modarchivemediaclass_defaults
       (void)
15 #else
16    VST_BOOLEAN
       vst_modarchivemediaclass_defaults
       ()
17 #endif
18 {
19    VST_BOOLEAN             rc =
       VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
       enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Modify Archive Media
       Class default parameters ***\n" );
29    vst_promptforglobals(&priority,
       user_field, &timeout, &retries,
       &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc =
       VSCMD_ModifyArchiveMediaClass_Set
       Defaults(
32       VSID_PRIORITY,          priority,
33       VSID_USER_FIELD,
       user_field,
34       VSID_TIMEOUT_VALUE,     timeout,
```

```
35       VSID_RETRY_LIMIT,       retries,
36       VSID_STATUS_WAIT_FLAG,
         wait_flag,
37       VSID_ENTERPRISE_ID,
         enterprise_id,
38       VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_ModifyArchiveMediaClass(l)

VSCMD_ ModifyMedia Class

`VSCMD_ModifyMediaClass` modifies the value of one or more attributes of a MediaClass group. A MediaClass group establishes common characteristics for the media it contains.

All MediaClass group attributes must be specified on a Modify Media Class request, whether the value of an attribute is being modified or not.

Synopsis

VST_BOOLEAN VSCMD_ModifyMediaClass
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The command handle for this Modify Media Class command.
- "…" = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value of the field to use for this request. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CAPACITY (VST_CAPACITY)	The maximum number of media allowed in this MediaClass group.

Parameter Type	Description
VSID_CLASS_MOUNT_STATE (VST_CLASS_MOUNT_STATE)	Indicates whether this MediaClass group supports the mount by MediaClass functionality. Valid VSID_CLASS_MOUNT_STATE values are enumerated in the *vs_types.h* file.
VSID_CLASS_RPC_OPTION (VST_CLASS_RPC_OPTION)	Indicates whether callbacks are activated for this MediaClass group and if they are, which callback scheme is used. Valid VSID_CLASS_RPC_OPTION values are enumerated in the *vs_types.h* file.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status for this request.
VSID_HIGH_MARK (VST_HIGH_MARK)	The percentage of VSID_CAPACITY above which the specified migration policy option is performed or initiated. This field is applicable only if VSID_ARCHIVE_ACTION is set to VSE_ARCHIVE_ACTION_NOTIFY or VSE_ARCHIVE_ACTION_MIG.
VSID_HOST_NAME (VST_HOST_NAME)	The network-assigned name of the computer where the task that listens for MediaClass callbacks executes. Applicable only if VSID_CLASS_RPB_OPTION is set to VSE_CLASS_RPC_STANDARD.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The unique name assigned to the MediaClass group.
VSID_NEW_MEDIA_CLASS_NAME (VST_NEW_MEDIA_CLASS_NAME)	The new, unique name assigned to the specified MediaClass group.

Parameter Type	Description
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	The media type supported by this MediaClass group. Valid media type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_NOTIFY_COMMENT (VST_NOTIFY_COMMENT)	The user-specified comment included in a system log message when the number of media in the MediaClass group exceeds the high mark threshold. The MediaClass name, fill level, high mark threshold, and capacity values are automatically included in the system log message and need not be included in `VSID_NOTIFY_COMMENT`.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Modify Media Class commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	The RPC procedure number of the client process to receive callbacks generated for this MediaClass group. `VSID_PROCEDURE_NUMBER` is required if `VSID_CLASS_RPC_OPTION` is set to `VSE_CLASS_RPC_ENTERPRISE`. Otherwise, `VSID_PROCEDURE_NUMBER` is not applicable.
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	The RPC program number of the client process to receive callbacks generated for this MediaClass group. `VSID_PROGRAM_NUMBER` is required if `VSID_CLASS_RPC_OPTION` is set to `VSE_CLASS_RPC_ENTERPRISE`. Otherwise, `VSID_PROGRAM_NUMBER` is not applicable.

Parameter Type	Description
VSID_PROTOCOL (VST_PROTOCOL)	The Internet protocol VolServ is to use to send callbacks for this MediaClass group. The default VSID_PROTOCOL is VSE_PROT_TCP. VSID_PROTOCOL is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_PROTOCOL is not applicable.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Modify Media Class commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Modify Media Class commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise to receive callbacks for this MediaClass group.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	A value to put in USER_FIELD for Modify Media Class commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Modify Media Class commands. Neither the API software nor VolServ uses USER_FIELD.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	The RPC version number of the client process to receive callbacks generated for this MediaClass group. VSID_VERSION_NUMBER is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_VERSION_NUMBER is not applicable.

Return Values

VSCMD_ModifyMediaClass returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode.
  - Good initial status received if the API is operating in asynchronous mode.
- VSE_FALSE - The command failed.
  A return code of VSE_FALSE (which is 0) means the command failed.
  To determine where the error occurred, and what the error was, the client queries the command's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.

  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ, and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API, and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /****************************************
       *********
2   *
```

```
3    * FUNCTION: vst_modmediaclass_execute
4    *
5    * PURPOSE:
6    * This executes the VSCMD_ModMediaClass
         API call.
7    *
8    * PARAMETERS:
9    * none
10   *
11 ************************************
         *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_modmediaclass_execute(void)
14 #else
15    VST_BOOLEAN
         vst_modmediaclass_execute()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_MEDIA_CLASS_NAME    mediaclass;
20    VST_MEDIA_CLASS_NAME    newclass;
21    VST_CAPACITY            capacity;
22    VST_CLASS_MOUNT_STATE   mountstate;
23    VST_HIGH_MARK           highmark;
24    VST_COMMAND_HANDLE      cmd;
25    VST_NOTIFY_COMMENT      comment;
26    VST_CLASS_RPC_OPTION    rpc_option;
27    VST_HOSTNAME
         rpc_hostname;
28    VST_PROGRAM_NUMBER      rpc_prognum;
29    VST_VERSION_NUMBER      rpc_versnum;
30    VST_PROCEDURE_NUMBER    rpc_procnum;
31    VST_PROTOCOL
         rpc_protocol;
32    VST_ENTERPRISE_ID
         enterpriseid;
33
34    /* get parameters from user */
35    printf("*** Modify Media Class
         parameters ***\n" );
```

```
36    printf("\nEnter Media Class to modify
         ==>");
37    gets( mediaclass);
38    printf("\nEnter New Media Class Name
         (return to leave unchanged) ==>");
39    gets( newclass);
40    printf("\nEnter capacity==>");
41    capacity = atoi(gets(input));
42    printf("\nEnter class mount state (0)
         no, (1) ok: ");
43    mountstate = atoi(gets(input));
44    printf("\nEnter high mark ==>");
45    highmark = atoi(gets(input));
46    printf("\nEnter notify comment
         ==>");
47    gets(comment);
48    printf("\nEnter Option (0) no
         callbacks, (1) standard, (2)
         Enterprise ==>");
49    rpc_option = (VST_CLASS_RPC_OPTION)
         atoi(gets(input));
50    if (rpc_option ==
         VSE_CLASS_RPC_NONE)
51    {
52
         VSCMD_ModifyMediaClass_SetDefault
         s(
53                VSID_CLASS_RPC_OPTION,
         rpc_option,
54                VSID_ENDFIELD);
55    }
56    else if (rpc_option ==
         VSE_CLASS_RPC_STANDARD)
57    {
58       printf("\nEnter RPC Host Name
         ==>");
59       gets(rpc_hostname);
60       printf("\nEnter program number
         ==>");
61       rpc_prognum =
         (VST_PROGRAM_NUMBER)
         atol(gets(input));
```

```
62        printf("\nEnter version number
          ==>");
63        rpc_versnum =
          (VST_PROGRAM_NUMBER)
          atol(gets(input));
64        printf("\nEnter procedure number
          ==>");
65        rpc_procnum =
          (VST_PROGRAM_NUMBER)
          atol(gets(input));
66        printf("\nEnter Protocol (6) TCP
          or (17) UDP ==>");
67        rpc_protocol = (VST_PROTOCOL)
          atoi(gets(input));
68
          VSCMD_ModifyMediaClass_SetDefault
          s(
69           VSID_HOST_NAME, rpc_hostname,
70           VSID_CLASS_RPC_OPTION,
          rpc_option,
71           VSID_PROGRAM_NUMBER,
          rpc_prognum,
72           VSID_VERSION_NUMBER,
          rpc_versnum,
73           VSID_PROCEDURE_NUMBER,
          rpc_procnum,
74           VSID_PROTOCOL,
          rpc_protocol,
75           VSID_ENDFIELD);
76     }
77     else if (rpc_option ==
          VSE_CLASS_RPC_ENTERPRISE)
78     {
79        printf("\nEnter enterprise id
          ==>");
80        enterpriseid =
          (VST_ENTERPRISE_ID)
          atol(gets(input));
81
          VSCMD_ModifyMediaClass_SetDefault
          s(
```

```
82          VSID_CLASS_RPC_OPTION,
          rpc_option,
83          VSID_TARGET_ENTERPRISE_ID,
          enterpriseid,
84          VSID_ENDFIELD);
85    }
86
87    /* create the command handle */
88    /* Note that the command handle is
          not */
89    /* destoyed in this routine, but in
          */
90    /* vst_dispatch when final status is
          received. */
91    cmd = VS_Command_Create();
92    if ( cmd != (VST_COMMAND_HANDLE)
          NULL)
93    {
94       /* Send the command to the VolServ
          software. */
95       /* Note that status is not
          processed here. */
96       /* Instead, it is processed in the
          */
97       /* vst_dispatch routine. Also,
          note that */
98       /* default values such as timeout
          */
99       /* value retry limit and priority
          are set as */
100      /* default parameters. */
101      rc = VSCMD_ModifyMediaClass(cmd,
102         VSID_MEDIA_CLASS_NAME,
          mediaclass,
103         VSID_NEW_MEDIA_CLASS_NAME,
          newclass,
104         VSID_CAPACITY,
          capacity,
105         VSID_CLASS_MOUNT_STATE,
          mountstate,
106         VSID_HIGH_MARK,
          highmark,
```

```
107         VSID_NOTIFY_COMMENT,
          comment,
108         VSID_ENDFIELD);
109   }
110   return ( rc );
111}
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generate intermediate status in response to a Modify Media Class request.

The Modify Media Class command triggers unsolicited MediaClass callbacks from VolServ.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

The name specified for the MediaClass group must be unique. Any requests to create non-unique MediaClass names fail.

MediaClass groups can span archives.

MediaClass groups may contain only one type of media.

Checks to determine if `VSID_HIGH_MARK` has been reached or exceeded occur after any Reclassify, Import, or Modify Media Class command.

`VSID_CAPACITY` is a hard limit. VolServ does not permit the number of media assigned to a MediaClass group to exceed the `VSID_CAPACITY` for that MediaClass group.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Modify Media Class request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Modify Media Class commands are set with `VSCMD_ModifyMediaClass_SetDefaults`. If command-specific defaults are set for Modify Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Modify Media Class command, the parameter identifier and the value used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Modify Media Class command:

- `VSID_MEDIA_CLASS_NAME`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_CreateMediaClass(l),
- VSCMD_DeleteMediaClass(l),
- VSCMD_ModifyMediaClass_SetDefaults(l)

VSCMD_ ModifyMedia Class_Set Defaults

`VSCMD_ModifyMediaClass_SetDefaults` sets the command-level default parameters for Modify Media Class commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Modify Media Class commands are set with `VSCMD_ModifyMediaClass_SetDefaults`. If command-specific defaults are set for Modify Media Class commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Modify Media Class command, the parameter identifier and the value used for the parameter can be submitted on the specific request itself.

Synopsis

VST_BOOLEAN VSCMD_ModifyMediaClass
(“…”,
VSID_ENDFIELD)

Arguments

- **”...”** = Variable length argument list consisting of pairs of arguments. Each pair of Arguments consists of a parameter identifier, followed by the value used as a command default

value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CAPACITY (VST_CAPACITY)	The maximum number of media allowed in this MediaClass group.
VSID_CLASS_MOUNT_STATE (VST_CLASS_MOUNT_STATE)	Indicates whether this MediaClass group supports the “mount by MediaClass” functionality. Valid VSID_CLASS_MOUNT_STATE values are enumerated in the *vs_types.h* file.
VSID_CLASS_RPC_OPTION (VST_CLASS_RPC_OPTION)	Indicates whether callbacks are activated for this MediaClass group and if they are, which callback scheme is used. Valid VSID_CLASS_RPC_OPTION values are enumerated in the *vs_types.h* file.
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (int)	The number of enterprise identifiers in the list.
(VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status for this request.
VSID_HIGH_MARK (VST_HIGH_MARK)	The percentage of VSID_CAPACITY above which the specified migration policy option is performed or initiated. This field is applicable only if VSID_ARCHIVE_ACTION is set to VSE_ARCHIVE_ACTION_NOTIFY or VSE_ARCHIVE_ACTION_MIG.

Parameter Type	Description
VSID_HOST_NAME (VST_HOST_NAME)	The network-assigned name of the computer where the task that listens for MediaClass callbacks executes. Applicable only if VSID_CLASS_RPB_OPTION is set to VSE_CLASS_RPC_STANDARD.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The current name assigned to the MediaClass group.
VSID_NEW_MEDIA_CLASS_NAME (VST_NEW_MEDIA_CLASS_NAME)	The current name assigned to the MediaClass group.
VSID_MEDIA_TYPE_NAME (VST_MEDIA_TYPE_NAME)	The media type supported by this MediaClass group. Valid media type names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_NOTIFY_COMMENT (VST_NOTIFY_COMMENT)	The user-specified comment included in a system log message when the number of media in the MediaClass group exceeds the high mark threshold. The MediaClass name, fill level, high mark threshold, and capacity values are automatically included in the system log message and need not be included in VSID_NOTIFY_COMMENT.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Modify Media Class commands. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_PROCEDURE_NUMBER (VST_PROCEDURE_NUMBER)	The RPC procedure number of the client process to receive callbacks generated for this MediaClass group. VSID_PROCEDURE_NUMBER is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_PROCEDURE_NUMBER is not applicable.

Parameter Type	Description
VSID_PROGRAM_NUMBER (VST_PROGRAM_NUMBER)	The RPC program number of the client process to receive callbacks generated for this MediaClass group. VSID_PROGRAM_NUMBER is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_PROGRAM_NUMBER is not applicable.
VSID_PROTOCOL (VST_PROTOCOL)	The Internet protocol VolServ is to use to send callbacks for this MediaClass group. The default VSID_PROTOCOL is VSE_PROT_TCP. VSID_PROTOCOL is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_PROTOCOL is not applicable.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Modify Media Class commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Modify Media Class commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise to receive callbacks for this MediaClass group.

Parameter Type	Description
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	A value to put in USER_FIELD for Modify Media Class commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Modify Media Class commands. Neither the API software nor VolServ uses USER_FIELD.
VSID_VERSION_NUMBER (VST_VERSION_NUMBER)	The RPC version number of the client process to receive callbacks generated for this MediaClass group. VSID_VERSION_NUMBER is required if VSID_CLASS_RPC_OPTION is set to VSE_CLASS_RPC_ENTERPRISE. Otherwise, VSID_VERSION_NUMBER is not applicable.

Return Values

VSCMD_ModifyMediaClass_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_modmediaclass_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_ModMediaClass API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_modmediaclass_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_modmediaclass_defaults()
17 #endif
18 {
19    VST_BOOLEAN rc = VSE_FALSE;
20    VST_PRIORITY priority;
21    VST_USER_FIELD user_field;
22    VST_TIME_OUT timeout;
23    VST_RETRY_LIMIT retries;
24    VST_STATUS_WAIT_FLAG wait_flag;
25    VST_ENTERPRISE_ID enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Create Archive Media
         Class default parameters ***\n" );
```

```
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc =
         VSCMD_ModifyMediaClass_SetDefault
         s(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```


Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_ModifyMediaClass(l)

VSCMD_ ModifyPool

VSCMD_ModifyPool modifies the list of drives associated with a drive pool. A client issues a VSCMD_ModifyPool request to modify the list of drives that are included in a drive pool description. The client can add drives to and/or delete drives from the specified drive pool with a single VSCMD_ModifyPool request.

Synopsis

VST_BOOLEAN VSCMD_ModifyPool
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- handle = The command handle for the Modify Pool request.
- "…" = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.
VSID_DRIVE_ID_LIST (int)	The number of drives added to or deleted from the specified drive pool.

Parameter Type	Description
(VST_DRIVE_ID)	The identifier of drives added to or deleted from the specified drive pool.
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	The name of the drive pool whose list of drives are modified. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_MODPOOL_OPTION_LIST (VST_POOL_DRIVE_OPTION)	A pointer to the list of actions that correspond to the drives identified in the `VSID_DRIVE_ID` table. An entry in the `VSID_MODPOOL_OPTION_LIST` indicates whether the corresponding entry in `VSID_DRIVE_ID_LIST` is added to or deleted from the specified drive pool. Valid `VSID_MODPOOL_OPTION` values are enumerated in the *vs_types*.h file.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_ModifyPool returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.

  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code's value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.

- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code's value is `VSE_VOLSERV`, the error occurred in VolServ and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code's value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NOTINITIALIZED` - The VolServ API is not initialized.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument.

- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_modpool_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_ModifyPool API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_modpool_execute(void)
14 #else
15    VST_BOOLEAN vst_modpool_execute()
16 #endif
17 {
18  VST_BOOLEAN            rc = VSE_FALSE;
19  VST_DRIVE_POOL_NAME    dp;
20  int                    count;
21  int                    i;
22  VST_DRIVE_ID
         drivelist[VST_MAX_ITEMS];
23  VST_POOL_DRIVE_OPTION
         optionlist[VST_MAX_ITEMS];
24  VST_COMMAND_HANDLE     cmd;
25
26    /* get parameters from user */
27    printf("*** Modify Pool Parameters
         ***\n" );
```

```
28    printf("\nEnter Drive Pool to modify
         ==>");
29    gets( dp );
30    count = vst_getdrivelist(drivelist,
         VST_MAX_ITEMS);
31    printf("\nPlease select the action to
         take on each drive\n");
32    for (i = 0; i < count; i++)
33    {
34       printf("drive %d: (1) add to pool,
         (2) delete: ", drivelist[i]);
35       optionlist[i] =
         (VST_POOL_DRIVE_OPTION)
         atoi(gets(input));
36    }
37
38    /* create the command handle */
39    /* Note that the command handle is
         not */
40    /* destoyed in this routine, but in
         */
41    /* vst_dispatch when final status is
         received. */
42    cmd = VS_Command_Create();
43    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
44    {
45       /* Send the command to the VolServ
         software. */
46       /* Note that status is not
         processed here. */
47       /* Instead, it is processed in the
         */
48       /* vst_dispatch routine. Also,
         note that */
49       /* default values such as timeout,
         */
50       /* value retry limit and priority
         are set as */
51       /* default parameters. */
52       rc = VSCMD_ModifyPool(cmd,
53          VSID_DRIVEPOOL_NAME, dp,
```

```
54          VSID_DRIVE_ID_LIST, count,
          drivelist,
55          VSID_MODPOOL_OPTION_LIST,
          count, optionlist,
56          VSID_ENDFIELD);
57     }
58     return ( rc );
59 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Modify Pool request.

`VSCMD_ModifyPool` does not trigger any MediaClass callbacks from VolServ.

The name specified for a new drive pool must be unique. A request to create a drive pool with a non-unique name fails.

Drive pools can contain zero or more drives.

Drives belonging to a single drive pool can be associated with different archives. Drives are not required to be associated with an archive to belong to a drive pool.

Drive pools can contain drives that support incompatible media types.

If a drive pool is specified on a Mount request and the specified drive pool spans archives, VolServ can select a drive to honor the Mount request that is in a different archive than the medium that is selected to honor the request. If this occurs, a Move-Mount action is required. If permitted, the medium is scheduled for ejection from its parent archive and eventually enters the archive associated with the assigned drive.

Whether or not Move-Mount action processing is permitted is specified at the archive level. The ACTION_MODE and MOVEWAIT_OPTION attributes control whether or not Move-Mount processing is allowed for a specific archive. These attributes are discussed under the VS_Archive_SetFields and VS_Archive_GetFields functions.

The VSID_DRIVE_ID_LIST and VSID_MODPOOL_OPTION_LIST parameters require that two arguments be passed instead of one.

- The first argument passed is the number of drives to add to or delete from the specified drive pool.
- The second argument is the list of identifier of the drives to add or delete from the specified drive pool.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive final status on a Modify Pool request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.

- Command-specific parameter defaults for Modify Pool commands are set with `VSCMD_ModifyPool_SetDefaults`. If command-specific defaults are set for Modify Pool commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Modify Pool command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Modify Pool request:

- `VSID_DRIVE_ID`,
- VSID_DRIVE_ID_ENTRY,
- VSID_DRIVE_ID_TABLE,
- `VSID_DRIVEPOOL_NAME`,
- `VSID_ERROR_CODE`,
- `VSID_ERROR_CODE_ENTRY`,
- VSID_ERROR_CODE_TABLE,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_CreatePool(l),
- VSCMD_DeletePool(l),
- VSCMD_ModifyPool_SetDefaults(l)

VSCMD_ ModifyPool_ SetDefaults

`VSCMD_ModifyPool_SetDefaults` sets the command-level default parameters for Modify Pool commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Modify Pool commands are set with `VSCMD_ModifyPool_SetDefaults`. If command-specific defaults are set for Modify Pool commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Modify Pool command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

VST_BOOLEAN VSCMD_ModifyPool_SetDefaults
(“…”,
VSID_ENDFIELD)

Arguments

- *“…”* = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on Modify Pool commands.
VSID_DRIVE_ID_LIST (int)	The number of drives to be added to or deleted from the specified drive pool.
(VST_DRIVE_ID *)	A pointer to the list of drives to be added to or deleted from the specified drive pool.
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	The name of the drive pool whose list of drives is modified. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_MODPOOL_OPTION_LIST (int)	The number of entries in the modify pool option list.
(VST_POOL_DRIVE_OPTION *)	A pointer to the list of actions that correspond to the drives identified in the VSID_DRIVE_ID table.
VSID_MODPOOL_OPTION (VST_POOL_DRIVE_OPTION)	An overall option to perform for all drives in VSID_DRIVE_ID_LIST. VSID_MODPOOL_OPTION replaces VSID_MODPOOL_OPTION_LIST when all drives are to be modified in some manner.

Parameter Type	Description
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Modify Pool commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Modify Pool commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Modify Pool commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for Modify Pool commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Modify Pool commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_ModifyPool_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_modpool_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_ModifyPool API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_modpool_defaults(void)
15 #else
16    VST_BOOLEAN vst_modpool_defaults()
17 #endif
18 {
19    VST_BOOLEAN            rc =
         VSE_FALSE;
```

```
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Modify Pool default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_ModifyPool_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```


Notes

The `VSID_DRIVE_ID_LIST` and `VSID_MODPOOL_OPTION_LIST` parameters require that two arguments be passed instead of one.

- The first argument passed is the number of drives to be added to or deleted from the specified drive pool.

- The second argument is the list of identifier of the drives to be added to or deleted from the specified drive pool.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_ModifyPool(l)

VSCMD_ Mount

VSCMD_Mount requests that VolServ mount a medium on a drive.

When issuing a Mount command, the client can specify:

- A single medium
- A list of media
- A MediaClass group
- and either:
  - A specific drive
  - A drive pool
  - A drive pool with the exclusion of drives

When a MediaClass group is specified, the client has the option to reclassify the selected medium into a different MediaClass group by specifying the VSID_TARGET_MEDIA_CLASS_NAME parameter. If this option is specified, VolServ reclassifies the selected medium before performing any other action on the medium. Using this option is similar to issuing a Reclassify request for the medium.

The Mount request supports a lock identifier parameter. This parameter is required if the drive to be used in satisfying the Mount request is locked. If a Mount request is issued for a locked drive and does not specify a lock identifier, the Mount request waits until the requested drive is unlocked.

The client also has the option of specifying how to handle a mount that requires an inter-archive movement of the medium. This option indicates whether an inter-archive medium movement should or should not be attempted and whether to consider if the source and destination archives are operator attended or not.

If more than one medium and/or more than one drive is specified on the Mount request, VolServ applies a selection algorithm to select a medium/drive pair from the lists of media and drives.

For manual archives, a Mount notice is sent to the operator console for action. The operator is then responsible for confirmation to VolServ when the mount is complete. VolServ returns status to the client after the operator action is complete.

A Mount request is queued for later processing if:

- Specified/selected drive is busy.
- Specified/selected medium is busy.
- Specified/selected drive is locked and a Lock identifier was not specified on the Mount request.

When a Mount request is queued for later processing, VolServ returns intermediate status to the client that indicates the reason the Mount request was queued.

Synopsis

```
VST_BOOLEAN VSCMD_Mount
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)
```

Arguments

- `handle` = The command handle for the Mount request.
- `"…"` = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.
VSID_CRITERIA_GROUP_HANDLE (int)	The index of a criteria group in the criteria group table to be used in selecting a medium to satisfy the Mount request. The valid range for VSID_CRITERIA_GROUP_HANDLE (int) is 0 to 3, inclusive.
(VST_CRITERIAGROUP_HANDLE)	A criteria group in the criteria group table to be used in selecting a medium to satisfy the Mount request. There can be 0 to 4 criteria groups defined for a Mount request.
VSID_DRIVE_EXCL_LIST (int)	The number of drives to exclude from the specified drive pool. VSID_DRIVE_EXCL_LIST is applicable only if the client is requesting a mount by drive pool.
(VST_DRIVE_ID *)	The list of drives to exclude from the specified drive pool. VSID_DRIVE_EXCL_LIST is applicable only if the client is requesting a mount by drive pool.
VSID_DRIVE_ID (VST_DRIVE_ID)	The identifier of the drive to mount. VSID_DRIVE_ID is applicable only if the client is specifying a specific drive to be mounted.

Parameter Type	Description
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	The name of the drive pool where a drive is selected to satisfy the Mount request. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted. `VSID_DRIVEPOOL_NAME` is applicable only if the client is requesting a mount by drive pool.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_LOCK_ID (VST_LOCK_ID)	The lock identifier associated with a locked drive that is specified or selected to honor the Mount request. A lock identifier is assigned to a drive with a `VSCMD_Lock` request.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The name of the MediaClass group where a medium is selected to honor the Mount request. `VSID_MEDIA_CLASS_NAME` is applicable only if the client is specifying a mount by MediaClass group.
VSID_MEDIA_ID (VST_MEDIA_ID)	The identifier of the medium to be mounted. `VSID_MEDIA_ID` is applicable only if the client is specifying a mount of a specific medium.
VSID_MEDIA_ID_LIST (int)	The number of media in the list of media where the medium to honor the Mount request is selected. `VSID_MEDIA_ID_LIST (int)` is applicable only if the client is specifying a mount from a media identifier list.
(char **)	The list of media where the medium to honor the Mount request is selected. `VSID_MEDIA_ID_LIST (int)` is applicable only if the client is specifying a mount from a media identifier list.

Parameter Type	Description
VSID_MOUNT_HANDLE (VST_MOUNT_HANDLE)	A handle that contains Mount command parameters. A client can set mount parameters in a mount handle with the `VS_Mount_SetFields` function. A mount handle can be passed in a Mount request instead of specifying the mount parameters on the Mount request itself.
VSID_MOUNT_OPTION (VST_MOUNT_OPTION)	A flag that indicates which mount processing options are in effect for the Mount command. Valid `VSID_MOUNT_OPTION` values are listed in the *vs_defs.h* file.
VSID_MOVEWAIT_OPTION (VST_MOVEWAIT_OPTION)	Indicates whether a Mount request should wait or fail if an inter-archive medium movement is required to complete the Mount request and either the source or target archive is unattended. Valid `VSID_MOVEWAIT_OPTION` values are enumerated in the *vs_types.h* file.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_MEDIA_CLASS_NAME (VST_TARGET_MEDIA_CLASS _NAME)	The name of the MediaClass group where the mounted medium is reclassified if the reclassify option is in effect for the Mount request.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Mount returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode

  - Good initial status received if the API is operating in asynchronous mode

- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mount_pool_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_Mount API call
       for drive
7  * pools.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *
13 *************************************
       *********/
14 #ifdef ANSI_C
15    VST_BOOLEAN
         vst_mount_pool_execute(void)
16 #else
17    VST_BOOLEAN vst_mount_pool_execute()
18 #endif
19 {
20  VST_BOOLEAN              rc =
       VSE_FALSE;
21  int                      i;
22  int                      entry;
23  VST_DRIVE_POOL_NAME      drivepool;
24  VST_DRIVE_ID
       excllist[VST_MAX_ITEMS];
25  int                      exclcount;
26  VST_MEDIA_ID             mediaid;
```

```
27  VST_LOCK_ID               lockid;
28  VST_CRITERIAGROUP_HANDLE  grouph;
29  VST_COMMAND_HANDLE        cmdh;
30
31     /* get parameters from user */
32    printf ( “Enter Drive Pool for mount
         ==> “ );
33    gets(drivepool);
34    printf(“\n*** Exclusion List
         ***\n”);
35    exclcount =
         vst_getdrivelist(excllist,
         VST_MAX_ITEMS);
36    printf ( “Enter Media ID for mounting
         ==> “ );
37    gets(mediaid);
38    printf ( “Mount by criteria (1) yes
         or (2) no ==> “ );
39    entry = atoi(gets(input));
40    if ( entry == 1 )
41    {
42       printf ( “Enter number of criteria
         groups ==> “ );
43       entry = atoi(gets(input));
44       for ( i = 0 ; i < entry ; i++ )
45       {
46          grouph =
         vst_create_mount_criteria();
47          if ( grouph !=
         (VST_CRITERIAGROUP_HANDLE) NULL )
48          {
49             /* It is easiest to set
         criteria */
50             /* groups in the default
         function as */
51             /* done here. This is
         particularly */
52             /* true if it is desired to
         use more */
53             /* than one criteria group.
         */
54             VSCMD_Mount_SetDefaults (
```

```
55
        VSID_CRITERIA_GROUP_HANDLE, i,
        grouph,
56                 VSID_ENDFIELD );
57           }
58        }
59     }
60     printf("Enter lock id ==> ");
61     lockid = atol(gets(input));
62
63     /* create the command (assume that
          the api is */
64     /* initialized) */
65     cmdh = VS_Command_Create();
66     if ( cmdh != (VST_COMMAND_HANDLE)
          NULL)
67     {
68        /* execute the mount command. */
69        /* if sync, we will wait for the
          mount to */
70        /* complete if async, we will
          leave once */
71        /* initial status has been
          returned */
72        rc = VSCMD_Mount ( cmdh,
73           VSID_DRIVEPOOL_NAME,
          drivepool,
74           VSID_DRIVE_EXCL_LIST,
          exclcount, excllist,
75           VSID_MEDIA_ID,
          mediaid,
76           VSID_LOCK_ID,
          lockid,
77           VSID_ENDFIELD );
78     }
79     else
80     {
81        rc = VSE_FALSE;
82     }
83     return ( rc );
84 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Mount request.

`VSCMD_Mount` does not trigger any MediaClass callbacks from VolServ.

Depending on the number of available drives and pending `Mount` requests, a specific Mount request can take a relatively long time to satisfy.

A drive that is specified in a Mount request may not be the ideal drive on which to mount the specified medium. It may take considerably longer to mount the medium onto the drive than if a drive pool had been specified.

The `VSID_MOVEWAIT_OPTION` option has no affect on a Mount request for a medium and drive associated with the same archive.

If the specified drive is locked, the appropriate lock identifier must be supplied if that drive is considered in the selection process.

If a specified or selected drive is locked and the Mount request does not specify a lock identifier, VolServ returns an intermediate status message with a “drive locked” indication. VolServ then waits for the drive to become unlocked to continue execution of the command.

If the Mount request specifies a MediaClass group or a list of media and the reclassify option is selected, the reclassify to a different MediaClass group occurs after a specific medium is selected to satisfy the `Mount` request. Only the selected medium is reclassified.

If the reclassify option is selected, the receiving MediaClass group is checked for compatible media type as well as having adequate capacity for another medium (i.e., fill level less than capacity). If either condition is not satisfied, the Mount request fails.

A pending Mount request (awaiting drive or medium) can be cancelled.

If no drive specified on a Mount request is on-line, the Mount request fails.

If a medium and/or drive is specified and either the medium or drive (or both) are presently in-use, the Mount request waits for resources and returns intermediate status indicating the reason for the delay.

When specifying a drive pool that contains drives that support different types of media, only those drives that support the media type of the media specified in the Mount request are considered for selection.

If a list of media specified in a Mount request contains media of more than one type, the request fails.

When a medium/drive pairing requires the medium be moved within a single archive system (such as cross-aisle or inter-manipulator unit) the mount may take a while to complete. The `VSID_MOVEWAIT_OPTION` setting does NOT apply to intra-archive system movement.

When a Mount request with groups of media and/or drives is submitted, VolServ attempts to select a drive/medium pair where the drive and medium are associated with the same archive. If there are multiple drive/medium pair candidates, VolServ selects a drive/medium pair from the archive with a free drive if, possible.

If there is no drive/medium pair associated with the same archive, VolServ then selects a drive/medium pair where the drive and medium are associated with different archives. If there are multiple drive/medium pair candidates, VolServ selects a drive/medium pair from the archive with a free drive, if possible. If all archives contain the same number of drives with no drives available, VolServ then selects a drive/medium pair from the archive with the largest number of media.

When specifying a mount by MediaClass group associated across more than one archive, no inter-archive medium/drive pairing is permitted. The medium selected from the MediaClass group must be in the same archive as the selected drive; otherwise the Mount request fails.

When a medium is ejected (as a result an Export or Checkout request), no check is made to determine if a Mount request exists in the queue for the ejected medium. As a result, the Mount request remains queued until a drive is freed. At that time, the Mount request fails because the medium is not available. In other words, the request queue is not checked for impact on pending requests each time a resource changes its availability and after a medium/drive pair is identified. VolServ does not attempt re-pairing based on changed availability of resources.

A queued Mount request (awaiting drive or medium) can be cancelled.

The `VSID_CRITERIA_GROUP_HANDLE` parameter require that two arguments be passed instead of one.

- The first argument passed is the number of criteria groups handles to be used in selecting the medium to be mounted.
- The second argument is the list of criteria group handles to be used in selecting the medium to be mounted.

The VSID_DRIVE_EXCL_LIST parameter require that two arguments be passed instead of one.

- The first argument passed is the number of drives to be excluded from the specified drive pool.
- The second argument is the list of the identifier of the drives to be excluded from the specified drive pool.

The VSID_MEDIA_ID_LIST parameter requires that two arguments be passed instead of one.

- The first argument passed is the number of media specified where the medium to honor the Mount request is selected.
- The second argument is the list of media where the medium to honor the Mount request is selected.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive final status on a Mount request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.

- Command-specific parameter defaults for Mount commands are set with `VSCMD_Mount_SetDefaults`. If command-specific defaults are set for Mount commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Mount command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Mount request:

- `VSID_DRIVE_ID`,
- `VSID_DRIVE_ID_ENTRY`,
- `VSID_DRIVE_ID_TABLE`,
- `VSID_MEDIA_ID`,
- `VSID_MEDIA_ID_ENTRY`,
- `VSID_MEDIA_ID_TABLE`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`,
- `VSID_WAIT_REASON`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Mount_Create(l),
- VS_Mount_Destroy(l)
- VS_Mount_GetFields(l),
- VS_Mount_SetFields(l),
- VS_Status_GetFields(l),
- VSCMD_Mount_SetDefaults(l)

VSCMD_ Mount_Set Defaults

`VSCMD_Mount_SetDefaults` sets the command-level default parameters for Mount commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Mount commands are set with `VSCMD_Mount_SetDefaults`. If command-specific defaults are set for Mount commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Mount command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

VST_BOOLEAN VSCMD_Mount_SetDefaults
(“…”,
VSID_ENDFIELD)

Arguments

- **”...”** = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on Mount commands.
VSID_CRITERIA_GROUP_HANDLE (int)	The index of a criteria group in the criteria group table to use in selecting a medium to satisfy the Mount request. The valid range for VSID_CRITERIA_GROUP_HANDLE (int) is 0 to 3, inclusive.
(VST_CRITERIAGROUP_HANDLE)	A criteria group in the criteria group table to use in selecting a medium to satisfy the Mount request. There can be 0 to 4 criteria groups defined for a Mount request.
VSID_DRIVE_EXCL_LIST (int)	The number of drives to exclude from the specified drive pool. VSID_DRIVE_EXCL_LIST is applicable only if the client is requesting a Mount by drive pool.
(VST_DRIVE_ID *)	The list of drives to exclude from the specified drive pool. VSID_DRIVE_EXCL_LIST is applicable only if the client is requesting a Mount by drive pool.
VSID_DRIVE_ID (VST_DRIVE_ID)	The identifier of the drive to mount. VSID_DRIVE_ID is applicable only if the client specifies a specific drive to be mounted.

Parameter Type	Description
VSID_DRIVEPOOL_NAME (VST_DRIVE_POOL_NAME)	The name of the drive pool where a drive is selected to satisfy the Mount request. Valid drive pool names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted. VSID_DRIVEPOOL_NAME is applicable only if the client is requesting a mount by drive pool.
(VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on Mount commands.
VSID_LOCK_ID (VST_LOCK_ID)	The lock identifier associated with a locked drive that is specified or selected to honor the Mount request. A lock identifier is assigned to a drive with a VSCMD_Lock request.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The name of the MediaClass group where a medium is selected to honor the Mount request. VSID_MEDIA_CLASS_NAME is applicable only if the client is specifying a mount by MediaClass group.
VSID_MEDIA_ID (VST_MEDIA_ID)	The identifier of the medium to be mounted. VSID_MEDIA_ID is applicable only if the client is specifying a mount of a specific medium.
VSID_MEDIA_ID_LIST (int)	The number of media in the list of media where the medium to honor the Mount request is selected. VSID_MEDIA_ID_LIST is applicable only if the client is specifying a mount from a media identifier list.
(char **)	The list of media where the medium to honor the Mount request is selected. VSID_MEDIA_ID_LIST is applicable only if the client is specifying a mount from a media identifier list.

Parameter Type	Description
VSID_MOUNT_HANDLE (VST_MOUNT_HANDLE)	A handle that contains Mount command parameters. A client can set mount parameters in a mount handle with the `VS_Mount_SetFields` function. A mount handle can be passed in a Mount request instead of specifying the mount parameters on the Mount request itself.
VSID_MOUNT_OPTION (VST_MOUNT_OPTION)	A flag that indicates which mount processing options are in effect for the Mount command. Valid `VSID_MOUNT_OPTION` values are listed in the *vs_defs.h* file.
VSID_MOVEWAIT_OPTION (VST_MOVEWAIT_OPTION)	Indicates whether a Mount request should wait or fail if an inter-archive medium movement is required to complete the Mount request and either the source or target archive is unattended. Valid `VSID_MOVEWAIT_OPTION` values are enumerated in the *vs_types.h* file.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Mount commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Mount commands. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Mount commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_MEDIA_CLASS_NAME (VST_TARGET_MEDIA_CLASS_NAME)	The name of the MediaClass group where the mounted medium is reclassified if the reclassify option is active for the Mount request.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for Mount commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Mount commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Mount_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.

- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_mount_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Mount API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN vst_mount_defaults(void)
15 #else
16    VST_BOOLEAN vst_mount_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Modify Pool default
         parameters ***\n" );
```

```
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_Mount_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```


Notes

The `VSID_CRITERIA_GROUP_HANDLE` parameter require that two arguments be passed instead of one.

- The first argument passed is the number of criteria groups handles to use in selecting the medium to mount.
- The second argument is the list of criteria group handles to use in selecting the medium to mount.

The `VSID_DRIVE_EXCL_LIST` parameter require that two arguments be passed instead of one.

- The first argument passed is the number of drives to exclude from the specified drive pool.
- The second argument is the list of the identifier of the drives to exclude from the specified drive pool.

The `VSID_MEDIA_ID_LIST` parameter require that two arguments be passed instead of one.

- The first argument passed is the number of media specified where the medium to honor the Mount request is selected.
- The second argument is the list of media where the medium to honor the Mount request is selected.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_CriteriaGroup_Create(l),
- VS_CriteriaGroup_Destroy(l),
- VS_CriteriaGroup_SetFields(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VS_Mount_Create(l),
- VS_Mount_Destroy(l),
- VS_Mount_SetFields(l),
- VSCMD_Mount(l)

VSCMD_Move

VSCMD_Move directs the movement of media from one archive to another. Inter-archive media movement requires operator intervention. An operator must eject media from their home archives and enter them into the specified target archive. The eject and enter functionalities are available from the appropriate archive's console display. The eject and enter functionalities are not available from the API.

Upon receipt of a Move request, VolServ verifies the specified media exist, the target archive supports the media type name of the specified media, and there exists an appropriate archive media class with the target archive. Only one target archive can be specified on a move request. The current archive of each specified medium is commanded to eject the medium. An Eject request, specifying the target archive, is displayed on the archive console of each source archive. The operator selects and manually ejects/removes the media on each eject list. After a medium is ejected from its home archive, the target archive displays a corresponding Enter request for that medium. The operator then manually enters the medium on the Enter list into the target archive.

When the VSID_MOVE_OPTION parameter is set to VSE_MOVE_WAIT, VolServ waits until processing of the Move command completes before returning status to the client. The client must monitor the media movement in some other manner (e.g., MediaClass callbacks) to know when the media are ejected from the home archives and entered into the target archive.

When the VSID_MOVE_OPTION parameter is set to VSE_MOVE_NOWAIT, VolServ returns a status code to the client after the specified media are placed on the ejection candidate list of the home archives.

When media are ejected from the home archive and entered into the target archive, VolServ generates MediaClass callbacks, if any of the moved media are associated with MediaClass groups that are configured to generate callbacks from VolServ.

The Move command can be used to place homeless media into an archive. A homeless medium is an intransit medium that has no pending movement activity.

Synopsis

VST_BOOLEAN VSCMD_Move
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- `handle` = The command handle for the Move request.
- `"…"` = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status for this request.

Parameter Type	Description
VSID_MEDIA_ID_LIST (int)	The number of media to move to the target archive.
(char **)	A list of the identifiers of the media to move to the target archive.
VSID_MOVE_OPTION (VST_MOVE_OPTION)	Indicates whether VolServ returns final status to the client as soon as the Move command begins execution or after the Move command completes execution. Valid `VSID_MOVE_OPTION` values are enumerated in the *vs_types.h* file.
VSID_MOVEWAIT_OPTION (VST_MOVEWAIT_OPTION)	Indicates whether a Move request waits or fails if either the source archive or target archive is unattended. The only valid `VSID_MOVEWAIT_OPTION` value is `VSE_MOVEWAIT_ATTENDED`. The other `VSID_MOVEWAIT_OPTION` values enumerated in the *vs_types.h* file have no effect on `VSCMD_Move`.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The name of the archive where the specified media are to be moved. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Move returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode

  - Good initial status received if the API is operating in asynchronous mode

- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_move_execute
4  *
5  * PURPOSE:
6  * This function sends a move command to
       the VolServ.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_move_execute( void )
14 #else
15    VST_BOOLEAN vst_move_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_BOOLEAN          done =
         VSE_FALSE;
20    int                  attended;
21    int                  wait;
22    int                  count;
23    char                 *
         medialist[VST_MAX_ITEMS];
24    VST_ARCHIVE_NAME     archive;
25    VST_COMMAND_HANDLE   cmd;
26
27    /* get parameters from user */
```

```
28    printf(“*** Move parameters ***\n” );
29    printf(“\nEnter Archive: “);
30    gets( archive);
31
32    /* If the next prompt is answered
         yes, the move */
33    /* will fail unless both archives are
         attended. */
34    printf(“\nMove only with Attended
         Archives (0) no, (1) yes: “);
35    attended = atoi(gets(input));
36
37    /* The wait parameter will cause the
         move to */
38    /* not return final status until the
         physical */
39    /* move is complete, or return
         immediately. */
40    printf(“\nWait for status until Move
         complete (0) no, (1) yes: “ );
41     wait = atoi(gets(input));
42    count = vst_getmedialist(medialist,
         VST_MAX_ITEMS);
43
44    /* create the command handle */
45    /* Note that the command handle is
         not */
46    /* destoyed in this routine, but in
         */
47    /* vst_dispatch when final status is
         received.*/
48    cmd = VS_Command_Create();
49    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
50    {
51       /* Send the command to the VolServ
         software. */
52       /* Note that status is not
         processed here. */
53       /* Instead, it is processed in the
         */
```

```
54       /* vst_dispatch routine. Also,
         note that */
55       /* default values such as timeout,
         */
56       /* value retry limit and priority
         are set as */
57       /* default parameters. */
58       rc = VSCMD_Move(cmd,
59          VSID_TARGET_ARCHIVE_NAME,
         archive,
60          VSID_MEDIA_ID_LIST, count,
         medialist,
61          VSID_MOVEWAIT_OPTION,
         (attended == 1 ?
         VSE_MOVEWAIT_ATTENDED :
         VSE_MOVEWAIT_YES ),
62          VSID_MOVE_OPTION,
         (wait == 1 ? VSE_MOVE_WAIT :
         VSE_MOVE_NOWAIT ),
63          VSID_ENDFIELD);
64    }
65    return ( rc );
66 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Move request.

`VSCMD_Move` triggers MediaClass callbacks from VolServ.

Operator intervention is required for inter-archive media movement.

A medium that is allocated to a Move request is not available for another allocation until the Move completes.

`VSCMD_Move` requests movement of media between archives, not within a single archive.

If the VSID_MOVE_OPTION is specified as VSE_MOVE_NOWAIT, the status returned to the client indicates only the initial validity of the Move request. Actual completion of the move can be traced only via MediaClass callback processing, media querying, or operator monitoring.

The VSID_MEDIA_ID_LIST parameter requires that two arguments be passed instead of one.

- The first argument passed is the number of media to move.
- The second argument is the list of identifiers of the media to move.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive final status on a Move request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.

- Command-specific parameter defaults for Move commands are set with `VSCMD_Move_SetDefaults`. If command-specific defaults are set for Move commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Move command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Move request:

- `VSID_ERROR_CODE`,
- `VSID_ERROR_CODE_ENTRY`,
- `VSID_ERROR_CODE_TABLE`,
- `VSID_MEDIA_ID`,
- `VSID_MEDIA_ID_ENTRY`,
- `VSID_MEDIA_ID_TABLE`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VSCMD_Move_SetDefaults(1),
- VS_Status_GetFields(l)

VSCMD_Move_SetDefaults

`VSCMD_Move_SetDefaults` sets the command-level default parameters for Move commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Move commands are set with `VSCMD_Move_SetDefaults`. If command-specific defaults are set for Move commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Move command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_Move_SetDefaults
( "…",
VSID_ENDFIELD)
```

Arguments

- **"..."** = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on Move commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on Move commands.
VSID_MEDIA_ID_LIST (int)	The number of media to move to the target archive.
(char **)	A list of the identifiers of the media to move to the target archive.
VSID_MOVE_OPTION (VST_MOVE_OPTION)	Indicates whether VolServ returns final status to the client as soon as the Move command begins execution or after the Move command completes execution. Valid VSID_MOVE_OPTION values are enumerated in the *vs_types.h* file.
VSID_MOVEWAIT_OPTION (VST_MOVEWAIT_OPTION)	Indicates whether a Move request waits or fails if either the source or target archive is unattended. The only valid VSID_MOVEWAIT_OPTION value is VSE_MOVEWAIT_ATTENDED. The other VSID_MOVEWAIT_OPTION values enumerated in the *vs_types.h* file have no effect on VSCMD_Move.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Move commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Move commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Move commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_ARCHIVE_NAME (VST_ARCHIVE_NAME)	The name of the archive where the specified media are moved. Valid archive names may contain up to 16 alphanumeric characters, including spaces. Leading and trailing spaces are not permitted.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for Move commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Move commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_Move_SetDefaults` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_move_defaults
4  *
5  * PURPOSE:
6  * This function sets default parameters
       for the move
7  * command.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN vst_move_defaults(void)
15 #else
16    VST_BOOLEAN vst_move_defaults()
17 #endif
18 {
19    VST_PRIORITY            priority;
20    VST_USER_FIELD          user_field;
21    VST_TIME_OUT            timeout;
```

Functins

```
22    VST_RETRY_LIMIT          retries;
23    VST_STATUS_WAIT_FLAG     wait_flag;
24    VST_ENTERPRISE_ID
         enterprise_id;
25    VST_BOOLEAN              rc;
26
27    /* get parameters from user */
28    printf("*** Move defaults ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    rc = VSCMD_Move_SetDefaults
         ( VSID_PRIORITY, priority,
31             VSID_USER_FIELD,
         user_field,
32             VSID_TIMEOUT_VALUE,
         timeout,
33             VSID_RETRY_LIMIT,
         retries,
34             VSID_STATUS_WAIT_FLAG,
         wait_flag,
35             VSID_ENTERPRISE_ID,
         enterprise_id,
36             VSID_ENDFIELD);
37    return(rc);
38 }
```

Notes

The `VSID_MEDIA_ID_LIST` parameter requires that two arguments be passed instead of one.

- The first argument passed is the number of media to move.
- The second argument is the list of identifiers of the media to move.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Move(l)

VSCMD_Multi Mount

VSCMD_MultiMount mounts one or more media with a single command. Up to eight mount requests can be specified on a MultiMount command. The media specified on a MultiMount command are mounted atomically.

Synopsis

```
VST_BOOLEAN VSCMD_MultiMount
(VST_COMMAND_HANDLE handle,
"…",
 VSID_ENDFIELD)
```

ARGUMENTS

- handle = The command handle for the MultiMount request.
- "..." = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on this request.

Parameter Type	Description
VSID_MOUNT_HANDLE (int)	The index of this mount handle in a mount handle table. The index of the first mount handle for a MultiMount request should be 0. Enter one `VSID_MOUNT_HANDLE (int, VST_MOUNT_HANDLE)` parameter pair for each medium to be mounted with this MultiMount request. Up to eight `VSID_MOUNT_HANDLE` parameter pairs may be specified on a single MultiMount command.
(VST_MOUNT_HANDLE)	The mount handle for this individual mount request. Enter one `VSID_MOUNT_HANDLE (int, VST_MOUNT_HANDLE)` parameter pair for each medium to be mounted with this MultiMount request. Up to eight `VSID_MOUNT_HANDLE` parameter pairs may be specified on a single MultiMount command.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_MultiMount returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.

  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_multimount_execute
4  *
5  * PURPOSE:
6  * This function will test the
       VSCMD_Multimount call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_multimount_execute(void)
14 #else
15    VST_BOOLEAN vst_multimount_execute()
16 #endif
17 {
18    int                 i;
19    int                 num;
20    VST_BOOLEAN         rc = VSE_FALSE;
21    VST_COMMAND_HANDLE  cmdh;
22    VST_MOUNT_HANDLE
         mounth[VSD_MAX_MOUNT_REQS];
23
24    /* get parameters from user */
25    printf("*** MultiMount Parameters
         ***\n");
26    printf("Enter the number of mount
         requests ==> " );
27    num = atoi(gets(input));
28
```

```
29    /* loop through the number of mount
         request */
30    for ( i = 0 ; i < num ; i++ )
31    {
32       /* Create a mount handle. */
33       /* Each mount handle stores a
         single mount */
34       /* request. The MultiMount command
         accepts */
35       /* multiple mount requests
         andexecutes them */
36       /* all as one operation. */
37       mounth[i] =
         vst_create_mount_handle();
38       if ( mounth[i] !=
         (VST_MOUNT_HANDLE) NULL )
39          {
40          /* add the mount request to the
         t */
41          /* multimount via the command
         */
42          /* default function */
43          VSCMD_MultiMount_SetDefaults (
44                   VSID_MOUNT_HANDLE, i,
         mounth[i],
45                   VSID_ENDFIELD );
46       }
47       else
48       {
49          rc = VSE_FALSE;
50          break;
51       }
52    }
53
54    if ( rc )
55    {
56    cmdh = VS_Command_Create();
57    if (cmdh != (VST_COMMAND_HANDLE)
         NULL)
58       {
59          /* execute the multimount
         command, note */
```

```
60          /* that all parameters have
           been set */
61          /* via default functions if
           sync, we will */
62          /* wait for all mounts to
           complete if */
63          /* async, we will leave once
           initial */
64          /* status has been returned */
65          rc = VSCMD_MultiMount ( cmdh,
66
           VSID_ENDFIELD );
67       }
68       else
69       {
70          rc = VSE_FALSE;
71       }
72    }
73
74    /* destroy the mount handles that
           contain the */
75    /* individual mount requests. */
76    for ( i = 0 ; i < num ; i++ )
77    {
78       VS_Mount_Destroy ( mounth[i] );
79    }
80    return ( rc );
81 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generates intermediate status in response to a MultiMount request for the following situations.

- A drive is reserved for an individual Mount request.
- A medium is mounted for an individual Mount request.
- An individual Mount request is waiting on a locked drive.

- An individual Mount request is waiting on a busy drive.
- An individual Mount request is waiting on a busy media.

Thus, for a successful MultiMount command with two Mount requests, the client can expect at least four intermediate statuses (two for the reserve of drives and two for the mount notification).

`VSCMD_MultiMount` triggers MediaClass callbacks from VolServ.

The `VSID_MOUNT_HANDLE` parameter require that two Arguments be passed instead of one.

- The first argument passed is the index of the entry in the Mount handle table.
- The second argument is the mount handle to be stored in the Mount handle table. The Mount handle table is created and used internally by the API software and is not visible to the client.

The MultiMount command does not restrict the Mount request in its use of mount parameters. All mount parameters are valid for each individual Mount request.

A MultiMount request is an “all-or-nothing” request. If an entire MultiMount request cannot be satisfied because of resource conflicts or limitations, the entire command fails.

Partial mounts can be done if one of the last Mount commands fails and the previous Mount commands have already completed. MultiMount does not dismount completed mounts if one mount fails while another succeeds.

A MultiMount command reserves resources to ensure deadlock avoidance. After all individual mounts are issued, the resources are freed.

A MultiMount command should not be used to batch multiple non-related Mount requests into a single command. The overhead of checking for resource contention, the presence of deadlock, and reserving drives is significant. It is recommended that individual Mount requests be issued for unrelated Mount requests.

Only one MultiMount request is processed at a time. If a MultiMount request is received by VolServ and there is already an active MultiMount request, the new request is queued until the active command completes reserving resources.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a MultiMount request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for MultiMount commands are set with `VSCMD_MultiMount_SetDefaults`. If

command-specific defaults are set for MultiMount commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a MultiMount command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful MultiMount request:

- `VSID_DRIVE_ID`,
- `VSID_DRIVE_ID_ENTRY`,
- `VSID_DRIVE_ID_TABLE`,
- `VSID_ERROR_CODE`,
- `VSID_ERROR_CODE_ENTRY`,
- `VSID_ERROR_CODE_TABLE`,
- `VSID_FIELD`,
- `VSID_MEDIA_ID`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`,
- `VSID_WAIT_REASON`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VSID_Mount_Create(l),
- VS_Mount_SetFields(l),
- VS_Status_GetFields(l),
- VSCMD_MultiMount_SetDefaults(l)

VSCMD_ MultiMount_ SetDefaults

`VSCMD_MultiMount_SetDefaults` sets the command-level default parameters for MultiMount commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for MultiMount commands are set with `VSCMD_MultiMount_SetDefaults`. If command-specific defaults are set for MultiMount commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a MultiMount command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

VST_BOOLEAN VSCMD_MultiMount_SetDefaults
(“…”,
VSID_ENDFIELD)

Arguments

- **“…”** = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on MultiMount commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on MultiMount commands.
VSID_MOUNT_HANDLE (int)	The index of this mount handle in a mount handle table. The index of the first mount handle for a MultiMount request should be 0. Enter one VSID_MOUNT_HANDLE (int, VST_MOUNT_HANDLE) parameter pair for each medium to be mounted with a MultiMount request. Up to eight VSID_MOUNT_HANDLE parameter pairs may be specified on a single MultiMount command.
(VST_MOUNT_HANDLE)	The mount handle for this individual Mount request. Enter one VSID_MOUNT_HANDLE (int, VST_MOUNT_HANDLE) parameter pair for each medium to be mounted with a MultiMount request. Up to eight VSID_MOUNT_HANDLE parameter pairs may be specified on a single MultiMount command.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for MultiMount commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for MultiMount commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for MultiMount commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for MultiMount commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for MultiMount commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_MultiMount_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /**************************************
       *********
2  *
3  * FUNCTION: vst_multimount_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Multimount API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 **************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_multimount_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_multimount_defaults()
17 #endif
18 {
```

```
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Multimount default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_MultiMount_SetDefaults(
32       VSID_PRIORITY,          priority,
33       VSID_USER_FIELD,
         user_field,
34       VSID_TIMEOUT_VALUE,     timeout,
35       VSID_RETRY_LIMIT,       retries,
36       VSID_STATUS_WAIT_FLAG,
         wait_flag,
37       VSID_ENTERPRISE_ID,
         enterprise_id,
38       VSID_ENDFIELD);
39
40    return ( rc );
41 }
```


Notes

The `VSID_MOUNT_HANDLE` parameter requires that two arguments be passed instead of one.

- The first argument passed is the index of the entry in the mount handle table.

- The second argument is the mount handle to be stored in the mount handle table. The mount handle table is created and used internally by the API software and is not visible to the client.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_MultiMount(l)

VSCMD_Ping

VSCMD_Ping checks the availability of VolServ. If VolServ responds to a VSCMD_Ping request, it can be assumed by the client that the VolServ is available and functioning.

The client is not required to issue a VSCMD_Ping request before sending other requests to the VolServ system.

Upon receiving a VSCMD_Ping request, VolServ immediately returns a VolServ process identifier number. No other processing is executed for a VSCMD_Ping command.

Synopsis

VST_BOOLEAN VSCMD_Ping
(VST_COMMAND_HANDLE handle)

Arguments

- handle = The command handle for this Ping request.

Return Values

VSCMD_Ping returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.

  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_ping
4  *
5  * PURPOSE:
6  * This function tests the VSCMD_Ping API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_ping( void )
14 #else
15    VST_BOOLEAN vst_ping()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_COMMAND_HANDLE   cmd;
20
21    printf("*** Pinging VolServ ***\n" );
22
23    /* create the command handle */
24    cmd = VS_Command_Create();
25    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
26    {
27       /* Send the command to the VolServ
         software. */
28       /* This will try to "ping" the
         VolServ */
29       /* software at the host name and
         program */
30       /* number set in VS_Initialize */
31       rc = VSCMD_Ping(cmd);
32       vst_print_command(cmd);
33       if (rc)
34       {
35          printf("The VolServ software
         is active.\n");
36       }
37       else
38       {
39          printf("The VolServ software
         is down.\n");
40       }
41    }
42    vst_print_error(VSG_Error);
43    return ( rc );
44 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate or final status in response to a Ping request.

`VSCMD_Ping` triggers no MediaClass callbacks from VolServ.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

The following fields can be retrieved from the status handle after a successful Ping request:

- `VSID_PID`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l)

VSCMD_ QueryMount

VSCMD_QueryMount determines which drives can be used in a subsequent Mount command for the specified medium. A Query Mount request returns the drives in the order of preference, based upon availability, proximity to the medium, usage time, and usage count.

Upon receipt of a Query Mount request, VolServ determines which archive contains the specified medium.

If the specified medium is not in an archive, a null list of drives is returned to the client.

If the medium is in an archive, VolServ determines which drives in that archive are suitable for mounting the specified medium. The list of suitable drives is returned to the client in the Query Mount status.

Synopsis

```
VST_BOOLEAN VSCMD_QueryMount
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)
```

Arguments

- `handle` = The command handle for this Query Mount request.
- `"…"` = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_MEDIA_ID (VST_MEDIA_ID)	The identifier of the medium for which a list of compatible drives is being requested.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are `VSE_TRUE` (API software waits for final status) and `VSE_FALSE` (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE.`

Parameter Type	Description
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_QueryMount returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_querymount_execute
4  *
5  * PURPOSE:
```

```
6  * This executes the VSCMD_QueryMount API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
       vst_querymount_execute(void)
14 #else
15    VST_BOOLEAN vst_querymount_execute()
16 #endif
17 {
18    VST_BOOLEAN             rc =
       VSE_FALSE;
19    VST_MEDIA_ID            mediaid;
20    VST_COMMAND_HANDLE      cmd;
21
22    /* get parameters from user */
23    printf("*** QueryMount parameters
       ***\n" );
24
25    printf("\nEnter media ID: ");
26    gets( mediaid);
27
28    /* create the command handle */
29    /* Note that the command handle is
       not */
30    /* destoyed in this routine, but in
       */
31    /* vst_dispatch when final status is
       received. */
32    cmd = VS_Command_Create();
33    if ( cmd != (VST_COMMAND_HANDLE)
       NULL)
34    {
35       /* Send the command to the VolServ
       software. */
36       /* Note that status is not
       processed here. */
```

```
37       /* Instead, it is processed in the
         */
38       /* vst_dispatch routine. Also,
         note that */
39       /* default values such as timeout,
         */
40       /* value retry limit and priority
         are set as */
41       /* default parameters. */
42       rc = VSCMD_QueryMount(cmd,
43
         VSID_MEDIA_ID, mediaid,
44
         VSID_ENDFIELD);
45    }
46    return ( rc );
47 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Query Mount request for the following situations.

`VSCMD_QueryMount` does not trigger any MediaClass callbacks from VolServ.

The drives identified in the status returned to the client are known to be suitable for mounting the specified medium. However, they may not be available for mounting.

Drives that are not in the on-line state are not considered suitable for mounting and are, therefore, not returned in the Query Mount status.

If a Query Mount request specifies a medium that is currently mounted, the Query Mount request fails with the error `VSE_VOLERR_MEDIA_MOUNTED`. However, the status still includes a list of drives suitable for the specified medium.

The ordering of the drives in the list returned to the client is based on the medium's current physical location. Drives that are not mounted are listed before drives that are mounted. Consequently, for a mounted medium, the drive on which the medium is currently mounted may not be the first drive in the returned list.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Query Mount request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Query Mount commands are set with `VSCMD_QueryMount_SetDefaults`. If

command-specific defaults are set for Query Mount commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Query Mount command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Query Mount request:

- `VSID_DRIVE_ID`,
- `VSID_DRIVE_ID_ENTRY`,
- `VSID_DRIVE_ID_TABLE`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`,
- `VSID_WAIT_REASON`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),

- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_QueryMount_SetDefaults(l)

VSCMD_ QueryMount_ SetDefaults

`VSCMD_QueryMount_SetDefaults` sets the command-level default parameters for Query Mount commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Query Mount commands are set with `VSCMD_QueryMount_SetDefaults`. If command-specific defaults are set for Query Mount commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Query Mount command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

VST_BOOLEAN VSCMD_QueryMount_SetDefaults
(“…”,
VSID_ENDFIELD)

Arguments

- **“...”** = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command

default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on Query Mount commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on Query Mount commands.
VSID_MEDIA_ID (VST_MEDIA_ID)	The identifier of the medium for which a list of compatible drives is being requested.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Query Mount commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Query Mount commands. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Query Mount commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for Query Mount commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Query Mount commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_QueryMount_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_querymount_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_QueryMount API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_querymount_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_querymount_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Query Mount default
         parameters ***\n" );
```

```
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_QueryMount_SetDefaults(
32          VSID_PRIORITY,
         priority,
33          VSID_USER_FIELD,
         user_field,
34          VSID_TIMEOUT_VALUE,
         timeout,
35          VSID_RETRY_LIMIT,
         retries,
36          VSID_STATUS_WAIT_FLAG,
         wait_flag,
37          VSID_ENTERPRISE_ID,
         enterprise_id,
38          VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_QueryMount(l)

VSCMD_ Reclassify

`VSCMD_Reclassify` changes the MediaClass group with which the specified media are associated.

Upon receipt of a `VSCMD_Reclassify` request, VolServ verifies that each specified media identifier references a media of the type supported by the target MediaClass group.

If all specified media are of the appropriate media type, VolServ verifies that the target MediaClass group is not already filled to capacity.

If the target MediaClass group is already filled to capacity, a `VSCMD_Reclassify` request fails and a failure return code is returned to the client.

If the target MediaClass group is not already filled to capacity, only the media it takes to reach the capacity are reclassified. Any remaining media specified in a `VSCMD_Reclassify` request are not reclassified, and a failure indicator is returned to the client.

Synopsis

```
VST_BOOLEAN VSCMD_Reclassify
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)
```

Arguments

- `handle` = The command handle for the Reclassify request.
- "…" = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_MEDIA_CLASS_LIST (int)	The list of MediaClass groups with which the specified media are currently associated. Used with the media to be reclassified reside in different MediaClass groups.
(char **)	A list of the MediaClass groups with which the specified media are currently associated.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The MediaClass name with which all the specified media are currently associated if all of the specified media are associated with the same MediaClass group.
VSID_MEDIA_ID_LIST (int)	The identifier of the number of media to reclassify.
(char **)	A list of the identifiers of the media to reclassify.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The name of the target MediaClass group with which the specified media are to be associated.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_Reclassify` returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- `VSE_FALSE` - The request failed. A return code of `VSE_FALSE` (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code's value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.
- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code's value is `VSE_VOLSERV`, the error occurred in VolServ and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code's value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.

- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_reclassify_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_Reclassify API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_reclassify_execute(void)
14 #else
15    VST_BOOLEAN vst_reclassify_execute()
16 #endif
17 {
18    VST_BOOLEAN             rc =
         VSE_FALSE;
19    VST_BOOLEAN             done =
         VSE_FALSE;
20    int                     i;
```

```
21    int                       count;
22    VST_BOOLEAN               single_class
         = VSE_TRUE;
23    VST_MEDIA_CLASS_NAME
         old_media_class;
24    VST_MEDIA_CLASS_NAME
         target_media_class;
25    char                      *
         medialist[VST_MAX_ITEMS];
26    char                      *
         mediaclasslist

         [VST_MAX_ITEMS];
27    VST_COMMAND_HANDLE        cmd;
28
29    /* get parameters from user */
30    printf("*** Reclassify parameters
         ***\n" );
31    count = vst_getmedialist(medialist,
         VST_MAX_ITEMS);
32    printf("\nEnter Target media class:
         ");
33    gets( target_media_class);
34    printf("\n Are media in same class
         (1) yes, (0) no? ");
35    single_class = (VST_BOOLEAN)
         atoi(gets(input));
36    if (single_class)
37    {
38       printf("\n Enter media class: ");
39       gets( old_media_class);
40    }
41    else
42    {
43       count =
         vst_getmediaclasslist(mediaclassl
         ist, VST_MAX_ITEMS);
44    }
45
46    /* create the command handle */
47    /* Note that the command handle is
         not */
```

```
48    /* destoyed in this routine, but in
         */
49    /* vst_dispatch when final status is
         received. */
50    cmd = VS_Command_Create();
51    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
52    {
53       /* Send the command to the VolServ
         software. */
54       /* Note that status is not
         processed here. */
55       /* Instead, it is processed in the
         */
56       /* vst_dispatch routine. Also,
         note that */
57       /* default values such as timeout
         */
58       /* value retry limit and priority
         are set as */
59       /* default parameters. */
60       if ( single_class )
61       {
62          /* all media are in the same
         source class */
63          rc = VSCMD_Reclassify(cmd,
64             VSID_MEDIA_CLASS_NAME,
         old_media_class,
65
         VSID_TARGET_MEDIA_CLASS_NAME,
               target_media_class,
66             VSID_MEDIA_ID_LIST, count,
         medialist,
67             VSID_ENDFIELD);
68       }
69       else
70       {
71          /* The media are in different
         mediaclass */
72          /* groups*/
73          rc = VSCMD_Reclassify(cmd,
```

```
74              VSID_MEDIA_CLASS_LIST,
          count,
                mediaclasslist,
75
          VSID_TARGET_MEDIA_CLASS_NAME,
                target_media_class,
76              VSID_MEDIA_ID_LIST, count,
          medialist,
77              VSID_ENDFIELD);
78          }
79      }
80      return ( rc );
81  }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ does not generate intermediate status in response to a Reclassify request.

There are two ways to specify the MediaClass groups with which the specified media are currently associated.

- If all of the specified media are associated with the same MediaClass group, use the `VSID_MEDIA_CLASS_NAME` parameter.
- If the specified media are associated with more than one MediaClass group, use the `VSID_MEDIA_CLASS_LIST` parameter. The `VSID_MEDIA_CLASS_LIST` must contain an entry for each medium specified in `VSID_MEDIA_ID_LIST` (the lists must contain the same number of entries.) Also, the entries in these lists are positional. For example, the first specified medium must be associated with the first specified MediaClass group; the second specified medium must be associated with the

second specified MediaClass group; and the n^{th} specified medium must be associated with the n^{th} specified MediaClass group.

Pending Mount requests are not affected by the reclassification of media.

If a medium to be reclassified is in an archive, the target MediaClass group with which the medium is associated must be associated with that archive.

If the capacity of the target MediaClass group would be exceeded by the reclassification, only as many media as necessary to reach capacity are reclassified. The Reclassify request for any remaining media fails.

The capacity of an archive media class is a soft limit. If the capacity of an archive media class is exceeded, the entire Reclassify request is processed unless the capacity of the associated MediaClass group is also reached. When the capacity of an archive media class is reached, applicable High Mark threshold processing is initiated.

If the target MediaClass group is not associated with any archive, the Reclassify request fails.

An attempt to reclassify a medium into the MediaClass group with which it is already associated returns an error.

If reclassifying a medium places the medium in a MediaClass group that does not have the medium's present location as a preferred location, the medium is not moved simply to place it into a preferred location. If the medium is mounted and then dismounted, or ejected and then entered, an attempt is made to place the medium in a preferred location as defined by the target MediaClass group.

A medium that does not reside in an archive can be reclassified.

The VSID_MEDIA_CLASS_LIST parameter require that two Arguments be passed instead of one.

- The first argument passed is the number of MediaClass groups contained in the list of MediaClass identifiers.
- The second argument is the list of MediaClass groups with which the specified media are currently associated.

The VSID_MEDIA_ID_LIST parameter require that two arguments be passed instead of one.

- The first argument passed is the number of media to reclassify.
- The second argument is the list of media to reclassify.

If the VSID_ENTERPRISE_ID parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (VSID_RETRY_LIMIT plus 1) multiplied by VSID_TIMEOUT_VALUE.

When the API software is operating in asynchronous mode, client software must call VS_Select to receive final status on a Reclassify request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using VS_Global_SetFields and VS_Global_GetFields function calls.

- Command-specific parameter defaults for Reclassify commands are set with `VSCMD_Reclassify_SetDefaults`. If command-specific defaults are set for Reclassify commands, they override the global defaults for all commands..

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Reclassify command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Reclassify request:

- `VSID_ERROR_CODE`,
- `VSID_ERROR_CODE_ENTRY`,
- `VSID_ERROR_CODE_TABLE`,
- `VSID_MEDIA_ID`,
- `VSID_MEDIA_ID_ENTRY`,
- `VSID_MEDIA_ID_TABLE`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,

- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VSCMD_Reclassify_SetDefaults(l)

VSCMD_ Reclassify_ SetDefaults

`VSCMD_Reclassify_SetDefaults` sets the command-level default parameters for Reclassify commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Reclassify commands are set with `VSCMD_Reclassify_SetDefaults`. If command-specific defaults are set for Reclassify commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Reclassify command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

VST_BOOLEAN VSCMD_Reclassify_SetDefaults
(“…”,
VSID_ENDFIELD)

Arguments

- **“…”** = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on Reclassify commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on Reclassify commands.
VSID_MEDIA_CLASS_LIST (int)	The number of MediaClass groups with which the specified media are currently associated. Used with the media to be reclassified reside in different archives.
(char **)	A list of the MediaClass groups with which the specified media are currently associated.
VSID_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The MediaClass group with which all the specified media are currently associated if all of the specified media are associated with the same MediaClass group.
VSID_MEDIA_ID_LIST (int)	The number of media to reclassify.
(char **)	A list of the identifiers of the media to reclassify.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Reclassify commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Reclassify commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Reclassify commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TARGET_MEDIA_CLASS_NAME (VST_MEDIA_CLASS_NAME)	The name of the target MediaClass group with which the specified media are to be associated.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for Reclassify commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Reclassify commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values VSCMD_Reclassify_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_Reclassify_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Reclassify API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_reclassify_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_reclassify_defaults()
17 #endif
18 {
19    VST_BOOLEAN            rc =
         VSE_FALSE;
20    VST_PRIORITY           priority;
```

```
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Reclassify default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_Reclassify_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39
40    return ( rc );
41 }
```


Notes

The `VSID_MEDIA_CLASS_LIST` parameter require that two arguments be passed instead of one.

- The first argument passed is the number of MediaClass groups contained in the list of MediaClass identifiers.
- The second argument is the list of MediaClass groups with which the specified media are currently associated.

The VSID_MEDIA_ID_LIST parameter require that two arguments be passed instead of one.

- The first argument passed is the number of media to reclassify.
- The second argument is the list of media to reclassify.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Reclassify(l)

VSCMD_ Reprioritize

`VSCMD_Reprioritize` changes the execution priority of a request.

The client must supply the request identifier and request type of the request to be reprioritized when issuing a `VSCMD_Reprioritize` request.

The request identifier and request type of the request to reprioritize can be specified on the `VSCMD_Reprioritize` request or can be within a command handle that is specified on the `VSCMD_Reprioritize` request.

Upon receipt of a `VSCMD_Reprioritize` request, VolServ changes the priority of the specified pending request to the new priority. VolServ then reorders its command queue to reflect the request's new priority.

Synopsis

```
VST_BOOLEAN VSCMD_Reprioritize
(VST_COMMAND_HANDLE handle,
"...",
VSID_ENDFIELD)
```

Arguments

- `handle` = The command handle for the Reprioritize request.
- `"..."` = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on Reclassify commands.
VSID_COMMAND_HANDLE (VST_COMMAND_HANDLE)	The command handle of the request to reprioritize. If `VSID_REQUEST_ID` and `VSID_REQUEST_TYPE` are specified, `VSID_COMMAND_HANDLE` is not applicable.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_NEW_PRIORITY (VST_PRIORITY)	The new execution priority to be assigned to the specified request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this Reprioritize request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_REQUEST_ID (VST_REQUEST_ID)	The identifier of the request to reprioritize. If `VSID_COMMAND_HANDLE` is specified, `VSID_REQUEST_ID` is not applicable.
VSID_REQUEST_TYPE (VST_REQUEST_ID)	The request type of the request to reprioritize. Valid values for this field are enumerated in the *vs_types.h* file. `VSID_COMMAND_HANDLE` is specified, `VSID_REQUEST_TYPE` is not applicable.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_Reprioritize` returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- `VSE_FALSE` - The request failed. A return code of `VSE_FALSE` (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with `VS_Error_GetFields`) to retrieve the error handle's object code.
  - If the object code's value is `VSE_NONE`, the client must query the global error code (`VSG_Error`) to determine where the error occurred.
- `VSE_ERR_BADHANDLE` - Specified handle was not a valid command handle.
- `VSE_ERR_NULLHANDLE` - Specified handle was a null pointer.
  - If the object code's value is `VSE_VOLSERV`, the error occurred in VolServ and the client uses `VST_ERROR_NUMCODE` to identify the specific error.
  - If the object code's value is not `VSE_VOLSERV` and is not `VSE_NONE`, the error occurred in the API and the client uses `VST_ERROR_CODE` to identify the specific error.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.

- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_reprioritize_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_Reprioritize
       API call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 *************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_reprioritize_execute(void)
14 #else
15    VST_BOOLEAN
         vst_reprioritize_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_REQUEST_ID       req;
20    VST_REQUEST_TYPE     c;
21    VST_PRIORITY         p;
```

```
22    VST_COMMAND_HANDLE   cmd;
23
24    /* get parameters from user */
25    printf("*** Reprioritize parameters
         ***\n" );
26    printf("Enter Request ID ==> " );
27    req = (VST_REQUEST_ID)
         atol(gets(input));
28    printf("Enter Command Request Type
         ==> " );
29    c = (VST_REQUEST_TYPE)
         atol(gets(input));
30    printf("Enter New Priority ==> " );
31    p = (VST_PRIORITY)
         atoi(gets(input));
32    /* create the command handle */
33    /* Note that the command handle is
         not */
34    /* destoyed in this routine, but in
         */
35    /* vst_dispatch when final status is
         received. */
36    cmd = VS_Command_Create();
37    if (cmd != (VST_COMMAND_HANDLE )NULL)
38    {
39       /* Send the command to the VolServ
         software. */
40       /* Note that status is not
         processed here. */
41       /* Instead, it is processed in the
         */
42       /* vst_dispatch routine. Also,
         note that */
43       /* default values such as timeout
         */
44       /* value retry limit and priority
         are set as */
45       /* default parameters. */
46       rc = VSCMD_Reprioritize(cmd,
47
         VSID_REQUEST_ID, req,
```

```
48
          VSID_REQUEST_TYPE, c,
49
          VSID_NEW_PRIORITY, p,
50
          VSID_ENDFIELD);
51     }
52     return ( rc );
53 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Reprioritize request for the following situations.

`VSCMD_Reprioritize` does not trigger any MediaClass callbacks from VolServ.

To ensure timely processing, a `VSCMD_Reprioritize` request is assigned execution priority 0 (zero) by VolServ.

A `VSCMD_Reprioritize` request cannot be cancelled.

The new priority specified in a `VSCMD_Reprioritize` request may be higher or lower than the current priority of the specified request.

If the request specified in a `VSCMD_Reprioritize` request is already processing when the `VSCMD_Reprioritize` request is received, VolServ still processes the `VSCMD_Reprioritize` request. If all work on the specified request has completed, the `VSCMD_Reprioritize` request functions as a noop. However, if there is additional processing to perform to complete the specified request, VolServ reprioritizes the remaining processing.

Any client can reprioritize any command as long as the request identifier and request type are known.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the intermediate and final status for this request is returned to the enterprise registered with VolServ.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Reprioritize request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Reprioritize commands are set with `VSCMD_Reprioritize_SetDefaults`. If command-specific defaults are set for Reprioritize commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Reprioritize command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Reprioritize request:

- `VSID_REQUEST_ID`,
- `VSID_SEQUENCE_NUM`,
- `VSID_SEQUENCE_TABLE`,
- `VSID_STATUS_CODE`,
- `VSID_STATUS_TYPE`,
- `VSID_USER_FIELD`.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_Reprioritize_SetDefaults(l)

VSCMD_ Reprioritize_ SetDefaults

`VSCMD_Reprioritize_SetDefaults` sets the command-level default parameters for Reprioritize commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Reprioritize commands are set with `VSCMD_Reprioritize_SetDefaults`. If command-specific defaults are set for Reprioritize commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Reprioritize command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

VST_BOOLEAN VSCMD_Reprioritize_SetDefaults
(“…”,
VSID_ENDFIELD)

Arguments

- **“…”** = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on Reprioritize commands.
VSID_COMMAND_HANDLE (VST_COMMAND_HANDLE)	The command handle of the request to reprioritize. If VSID_REQUEST_ID and VSID_REQUEST_TYPE are specified, VSID_COMMAND_HANDLE is not applicable.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on Reprioritize commands.
VSID_NEW_PRIORITY (VST_PRIORITY)	The new execution priority to be assigned to the specified request. Assignable priority values are restricted to a range from 1 (highest) to 32 (lowest) inclusive.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Reprioritize commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_REQUEST_ID (VST_REQUEST_ID)	The identifier of the request to reprioritize. If VSID_COMMAND_HANDLE is specified, VSID_REQUEST_ID is not applicable.
VSID_REQUEST_TYPE (VST_REQUEST_ID)	The request type of the request to reprioritize. Valid values for this field are enumerated in the *vs_types.h* file. VSID_COMMAND_HANDLE is specified, VSID_REQUEST_TYPE is not applicable.

Parameter Type	Description
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Reprioritize commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Reprioritize commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for Reprioritize commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Reprioritize commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

`VSCMD_Reprioritize_SetDefaults` returns:

- `VSE_TRUE` - Successful execution.
- `VSE_FALSE` - API failure - An appropriate error code is set in `VSG_Error`.
- `VSE_ERR_BADFIELD` - An invalid parameter was specified.
- `VSE_ERR_BADSIZE` - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- `VSE_ERR_NULLSTRING` - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_reprioritize_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Reprioritize API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_reprioritize_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_reprioritize_defaults()
17 #endif
18 {
```

```
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Reprioritize default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_Reprioritize_SetDefaults(
32          VSID_PRIORITY,
         priority,
33          VSID_USER_FIELD,
         user_field,
34          VSID_TIMEOUT_VALUE,
         timeout,
35          VSID_RETRY_LIMIT,
         retries,
36          VSID_STATUS_WAIT_FLAG,
         wait_flag,
37          VSID_ENTERPRISE_ID,
         enterprise_id,
38          VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Reprioritize(l)

VSCMD_ Request-Query

`VSCMD_RequestQuery` requests information about an outstanding VolServ request. To execute a valid Request Query command, the client must provide the VolServ-assigned request identifier of the request for which information is needed.

Synopsis

```
VST_BOOLEAN VSCMD_RequestQuery
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)
```

Arguments

- `handle` = The command handle for the Request Query request.
- `"..."` = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on this request.

Parameter Type	Description
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_REQUEST_ID (VST_REQUEST_ID)	The VolServ-assigned identifier of the request to be queried. A valid request identifier must be specified in the YYYY:DD:MM format where YYYY represents the year, DD represents the day, and MM is the month.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.

Parameter Type	Description
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_RequestQuery returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.
- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.

- If the object code's value is VSE_VOLSERV, the error occurred in VolServ and the client uses VST_ERROR_NUMCODE to identify the specific error.
- If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API and the client uses VST_ERROR_CODE to identify the specific error.

- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_requestquery_execute
4  *
5  * PURPOSE:
6  * This executes the VSCMD_RequestQuery
       API call.
7  *
8  * PARAMETERS:
9  * none
10 *
```

```
11 ************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN
         vst_requestquery_execute(void)
14 #else
15    VST_BOOLEAN
         vst_requestquery_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    VST_REQUEST_ID       requestid;
20    VST_COMMAND_HANDLE   cmd;
21
22    /* get parameters from user */
23    printf("*** Request Query parameters
         ***\n" );
24    printf("Request ID to query: ");
25    requestid = (VST_REQUEST_ID)
         atoi(gets(input));
26
27    /* create the command handle */
28    /* Note that the command handle is
         not */
29    /* destoyed in this routine, but in
         */
30    /* vst_dispatch when final status is
         received. */
31    cmd = VS_Command_Create();
32    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
33    {
34       /* Send the command to the VolServ
         software. */
35       /* Note that status is not
         processed here. */
36       /* Instead, it is processed in the
         */
37       /* vst_dispatch routine. Also,
         note that */
38       /* default values such as timeout
         */
```

```
39       /* value retry limit and priority
         are set as */
40       /* default parameters. */
41       rc = VSCMD_RequestQuery(cmd,
42
         VSID_REQUEST_ID, requestid,
43                          VSID_ENDFIELD);
44    }
45    return ( rc );
46 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ generates no intermediate status in response to a Request Query request.

`VSCMD_RequestQuery` does not trigger any MediaClass callbacks from VolServ.

The request identifier as shown in *syslogs* and *vsadm* is not in the correct format for a `VSCMD_RequestQuery` request. The request identifier obtained from these sources has the format: 93:123:45678. A request identifier in this format must be converted to the ydddnnnnn format before being used as a parameter on a `VSCMD_RequestQuery` request. The request identifier 93:123:45678, converted to the appropriate format, is 312345678.

The client must specify the identifier of the request to query.

After a request completes processing, there is a relatively short period of time that the request shows a state of complete. Afterwards, all knowledge of the request is removed from the VolServ system and a subsequent `VSCMD_RequestQuery` request for that request fails.

Only one request can be queried per `VSCMD_RequestQuery` request.

If the `VSID_ENTERPRISE_ID` parameter is set to any value other than zero, the final status for this request is returned to the enterprise registered with VolServ.

The total length of time the API software waits for a command status in synchronous mode from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive final status on a Request Query request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Request Query commands are set with `VSCMD_RequestQuery_SetDefaults`. If command-specific defaults are set for Request Query commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Request Query command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Request Query request:

- VSID_ERROR_CODE,
- VSID_ERROR_CODE_ENTRY,
- VSID_ERROR_CODE_TABLE,
- VSID_REQUEST_HANDLE,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_RequestQuery_SetDefaults(l)

VSCMD_ Request Query_Set Defaults

`VSCMD_RequestQuery_SetDefaults` sets the command-level default parameters for Request Query commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Request Query commands are set with `VSCMD_RequestQuery_SetDefaults`. If command-specific defaults are set for Request Query commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of a Request Query command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Synopsis

```
VST_BOOLEAN VSCMD_RequestQuery_SetDefaults
( "…",
VSID_ENDFIELD)
```

Arguments

- **"..."** = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command

default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- `VSID_ENDFIELD` = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on Request Query commands.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on Request Query commands.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Request Query commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_REQUEST_ID (VST_REQUEST_ID)	The VolServ-assigned identifier of the request to be queried. A valid request identifier must be specified in theYYYY:DD:MM format where YYYY represents the year, DD represents the day, and MM is the month.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Request Query commands. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Request Query commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for Request Query commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Request Query commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_RequestQuery_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_requestquery_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_RequestQuery API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_requestquery_defaults(void)
15 #else
16    VST_BOOLEAN
         vst_requestquery_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Modify Pool default
         parameters ***\n" );
```

```
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
30    /* set the default parameters */
31    rc = VSCMD_RequestQuery_SetDefaults(
32       VSID_PRIORITY,          priority,
33       VSID_USER_FIELD,
         user_field,
34       VSID_TIMEOUT_VALUE,     timeout,
35       VSID_RETRY_LIMIT,       retries,
36       VSID_STATUS_WAIT_FLAG,
         wait_flag,
37       VSID_ENTERPRISE_ID,
         enterprise_id,
38       VSID_ENDFIELD);
39    return ( rc );
40 }
```

Notes

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_RequestQuery(l)

VSCMD_ Unlock

VSCMD_Unlock releases exclusive use of a drive or a set of drives. The drive to be unlocked and the assigned lock identifier for the drive must be specified.

A set of drives locked with a single VSCMD_Lock request can be unlocked with multiple VSCMD_Unlock commands.

A VSCMD_Unlock request can specify a subset of the drives locked for the issuing client. VolServ unlocks only those drives specified in the VSCMD_Unlock request.

Synopsis

VST_BOOLEAN VSCMD_Unlock
(VST_COMMAND_HANDLE handle,
"…",
VSID_ENDFIELD)

Arguments

- handle = The command handle for the Unlock request.
- "…" = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.
- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status for this request.

Parameter Type	Description
VSID_DRIVE_ID (int)	The number of drives to unlock.
(VST_DRIVE_ID *)	A pointer to the list of identifiers of the drives to unlock.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on this request.
VSID_LOCK_ID (VST_LOCK_ID)	The lock identifier assigned to the drives to unlock. A lock identifier is assigned to a drive when the drive is locked with the `VSCMD_Lock` command.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for this request. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for this request. `VSID_RETRY_LIMIT` is not applicable when the API software executes in asynchronous mode.
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for this request. Valid options are `VSE_TRUE` (API software waits for final status) and `VSE_FALSE` (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (`VSE_TRUE`) or in asynchronous mode (`VSE_FALSE`). The default `VSID_STATUS_WAIT_FLAG` value is `VSE_TRUE.`

Parameter Type	Description
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software for this request. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for this request. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for this request. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Unlock returns:

- VSE_TRUE
  - Successful execution if the API is operating in synchronous mode
  - Good initial status received if the API is operating in asynchronous mode
- VSE_FALSE - The request failed. A return code of VSE_FALSE (which is 0) means the request failed.
  - To determine where the error occurred, and what the error was, the client queries the request's error handle (with VS_Error_GetFields) to retrieve the error handle's object code.
  - If the object code's value is VSE_NONE, the client must query the global error code (VSG_Error) to determine where the error occurred.

- VSE_ERR_BADHANDLE - Specified handle was not a valid command handle.
- VSE_ERR_NULLHANDLE - Specified handle was a null pointer.
  - If the object code's value is VSE_VOLSERV, the error occurred in VolServ and the client uses VST_ERROR_NUMCODE to identify the specific error.
  - If the object code's value is not VSE_VOLSERV and is not VSE_NONE, the error occurred in the API and the client uses VST_ERROR_CODE to identify the specific error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.
- VSE_ERR_NOTINITIALIZED - The VolServ API is not initialized.
- VSE_ERR_NULLSTRING - A null value was passed to a string argument.
- VSE_ERR_SEND - The API software could not send the command request to VolServ. This may be an RPC communication error and can indicate VolServ is not executing.

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_unlock_execute
4  *
5  * PURPOSE:
```

```
6  * This executes the VSCMD_Unlock API
       call.
7  *
8  * PARAMETERS:
9  * none
10 *
11 ****************************************
       *********/
12 #ifdef ANSI_C
13    VST_BOOLEAN vst_unlock_execute(void)
14 #else
15    VST_BOOLEAN vst_unlock_execute()
16 #endif
17 {
18    VST_BOOLEAN          rc = VSE_FALSE;
19    int                  count;
20    VST_DRIVE_ID
         drivelist[VST_MAX_ITEMS];
21    VST_LOCK_ID          lockid;
22    VST_COMMAND_HANDLE   cmd;
23
24    /* get parameters from user */
25    printf("*** Unlock Parameters ***\n"
         );
26    count = vst_getdrivelist(drivelist,
         VST_MAX_ITEMS);
27    printf("\nEnter Lock ID ==>");
28    lockid = (VST_LOCK_ID)
         atoi(gets(input));
29
30    /* create the command handle */
31    /* Note that the command handle is
         not */
32    /* destoyed in this routine, but in
         */
33    /* vst_dispatch when final status is
         received. */
34    cmd = VS_Command_Create();
35    if ( cmd != (VST_COMMAND_HANDLE)
         NULL)
36    {
```

```
37       /* Send the command to the VolServ
         software. */
38       /* Note that status is not
         processed here. */
39       /* Instead, it is processed in the
         */
40       /* vst_dispatch routine. Also,
         note that */
41       /* default values such as timeout
         */
42       /* value retry limit and priority
         are set as */
43       /* default parameters. */
44       rc = VSCMD_Unlock(cmd,
45             VSID_DRIVE_ID_LIST,
         count,drivelist,
46             VSID_LOCK_ID,
         lockid,
47             VSID_ENDFIELD);
48    }
49    return ( rc );
50 }
```

Notes

The API must be initialized with a call to `VS_Initialize` before this function can be executed.

VolServ can generate intermediate status in response ton Unlock request.

`VSCMD_Unlock` does not trigger MediaClass callbacks from VolServ.

Drives specified on a `VSCMD_Unlock` request that are either not locked or that have a lock identifier different from the one specified on the `VSCMD_Unlock` request return a failure status.

The `VSID_DRIVE_ID` parameter requires that two arguments be passed instead of one.

- The first argument passed is the number of drives to unlock.
- The second argument is the list of identifiers of the drives to unlock.

The total length of time the API software waits for a command status, in synchronous mode, from VolServ is (`VSID_RETRY_LIMIT` plus 1) multiplied by `VSID_TIMEOUT_VALUE`.

When the API software is operating in asynchronous mode, client software must call `VS_Select` to receive intermediate and final status on an Unlock request submitted through the API interface to the VolServ system.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Unlock commands are set with `VSCMD_Unlock_SetDefaults`. If command-specific defaults are set for Unlock commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Unlock command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

The following fields can be retrieved from the status handle after a successful Unlock request:

- `VSID_DRIVE_ID`,

- VSID_DRIVE_ID_ENTRY,
- VSID_DRIVE_ID_TABLE,
- VSID_ERROR_CODE,
- VSID_ERROR_CODE_ENTRY,
- VSID_ERROR_CODE_TABLE,
- VSID_LOCK_ID,
- VSID_SEQUENCE_NUM,
- VSID_SEQUENCE_TABLE,
- VSID_STATUS_CODE,
- VSID_STATUS_TYPE,
- VSID_USER_FIELD.

Note

If the argument list does not end with VSID_ENDFIELD, unpredictable results occur.

See Also

- vsapi(l),
- VS_Command_Create(l),
- VS_Command_Destroy(l),
- VS_Error_GetFields(l),
- VS_Initialize(l),
- VS_Status_GetFields(l),
- VSCMD_Lock(l),
- VSCMD_Unlock_SetDefaults(l)

VSCMD_ Unlock_Set Defaults

`VSCMD_Unlock_SetDefaults` sets the command-level default parameters for Unlock commands.

Two levels of default parameter settings are used in the API software—global defaults and command-specific defaults.

- Global defaults for all commands are initialized at startup and can be set or retrieved using `VS_Global_SetFields` and `VS_Global_GetFields` function calls.
- Command-specific parameter defaults for Unlock commands are set with `VSCMD_Unlock_SetDefaults`. If command-specific defaults are set for Unlock commands, they override the global defaults for all commands.

Tip

To override a default (global or command-specific) parameter value for a specific instance of an Unlock command, the parameter identifier and the value to be used for the parameter can be submitted on the specific request itself.

Functins

Synopsis

VST_BOOLEAN VSCMD_Unlock_SetDefaults
(“…”,
VSID_ENDFIELD)

Arguments

- **“…”** = Variable length argument list consisting of pairs of Arguments. Each pair of Arguments consists of a parameter identifier, followed by the value to be used as a command default value for the field. The valid parameter identifiers and types for this function are shown in the following "Parameters" paragraph.

- VSID_ENDFIELD = Required at the end of the variable length argument list to indicate the end of the list.

Parameters

Parameter Type	Description
VSID_CLIENT_DISPATCH (VST_CLIENT_DISPATCH)	The name of the client dispatch routine to receive status on Unlock commands.
VSID_DRIVE_ID (int)	The number of drives to unlock.
(VST_DRIVE_ID *)	A pointer to the list of identifiers of the drives to unlock.
VSID_ENTERPRISE_ID (VST_ENTERPRISE_ID)	The identifier of the enterprise, if any, to receive intermediate and final status on Unlock commands.
VSID_LOCK_ID (VST_LOCK_ID)	The lock identifier assigned to the drives to unlock. A lock identifier is assigned to a drive when the drive is locked with the VSCMD_Lock command.
VSID_PRIORITY (VST_PRIORITY)	The requested execution priority for Unlock commands. Assignable priority values are restricted to the range from 1 (highest) to 32 (lowest) inclusive. The default priority value is 15.
VSID_RETRY_LIMIT (VST_RETRY_LIMIT)	The number of times the API software retries for command status from VolServ before returning a time-out to the client software for Unlock commands. VSID_RETRY_LIMIT is not applicable when the API software executes in asynchronous mode. The default retry limit is 3.

Parameter Type	Description
VSID_STATUS_WAIT_FLAG (VST_STATUS_WAIT_FLAG)	A flag indicating whether the API software waits for final status from VolServ (or times-out) for Unlock commands. Valid options are VSE_TRUE (API software waits for final status) and VSE_FALSE (API software does not wait for final status). Also determines whether the API software operates in synchronous mode (VSE_TRUE) or in asynchronous mode (VSE_FALSE). The default VSID_STATUS_WAIT_FLAG value is VSE_TRUE.
VSID_TIMEOUT_VALUE (VST_TIME_OUT)	The amount of time (in seconds) the API software waits for status from VolServ before returning a time-out to the client software. The default time-out value is 120 seconds.
VSID_USER_FIELD (VST_USER_FIELD)	The value to put in USER_FIELD for Unlock commands. USER_FIELD is a 16-character field provided for user information. Information entered in this field is echoed back to the user in every status message returned for Unlock commands. Neither the API software nor VolServ uses USER_FIELD.

Return Values

VSCMD_Unlock_SetDefaults returns:

- VSE_TRUE - Successful execution.
- VSE_FALSE - API failure - An appropriate error code is set in VSG_Error.
- VSE_ERR_BADFIELD - An invalid parameter was specified.
- VSE_ERR_BADSIZE - The value passed for a string parameter exceeds the maximum allowable length for that parameter.

- VSE_ERR_NULLSTRING - A null value was passed to a string argument

Example

```
1  /*************************************
       *********
2  *
3  * FUNCTION: vst_unlock_defaults
4  *
5  * PURPOSE:
6  * This function sets the default
       parameters for the
7  * VSCMD_Unlock API call.
8  *
9  * PARAMETERS:
10 * none
11 *
12 *************************************
       *********/
13 #ifdef ANSI_C
14    VST_BOOLEAN
         vst_unlock_defaults(void)
15 #else
16    VST_BOOLEAN vst_unlock_defaults()
17 #endif
18 {
19    VST_BOOLEAN             rc =
         VSE_FALSE;
20    VST_PRIORITY            priority;
21    VST_USER_FIELD          user_field;
22    VST_TIME_OUT            timeout;
23    VST_RETRY_LIMIT         retries;
24    VST_STATUS_WAIT_FLAG    wait_flag;
25    VST_ENTERPRISE_ID
         enterprise_id;
26
27    /* get parameters from user */
28    printf("*** Unlock default
         parameters ***\n" );
29    vst_promptforglobals(&priority,
         user_field, &timeout, &retries,
         &wait_flag, &enterprise_id);
```

```
30    /* set the default parameters */
31    rc = VSCMD_Unlock_SetDefaults(
32             VSID_PRIORITY,
         priority,
33             VSID_USER_FIELD,
         user_field,
34             VSID_TIMEOUT_VALUE,
         timeout,
35             VSID_RETRY_LIMIT,
         retries,
36             VSID_STATUS_WAIT_FLAG,
         wait_flag,
37             VSID_ENTERPRISE_ID,
         enterprise_id,
38             VSID_ENDFIELD);
39    return ( rc );
40 }
```


Notes

The `VSID_DRIVE_ID` parameter requires that two arguments be passed instead of one.

- The first argument passed is the number of drives to unlock.
- The second argument is the list of identifiers of the drives to unlock.

Note

If the argument list does not end with `VSID_ENDFIELD`, unpredictable results occur.

See Also

- vsapi(l),
- VS_Error_GetFields(l),
- VS_Global_SetFields(l),
- VSCMD_Unlock(l)

NOTES

NOTES

A

Valid Status Fields

Roadmap

Topic	Refer To Chapter
Naming conventions. Global parameters. Error handling.	1
API functions.	2
Valid staus fields.	A
Error codes.	B
Mount example.	C

Table A-1

VSID_ACTION_CODE through VSID_COMPONENT_HANDLE_ENTRY

	VSID_ ACTION_ CODE	VSID_ ACTION_ CODE_ENTRY	VSID_ ACTION_ CODE_TABLE	VSID_ ARCHIVE_ HANDLE	VSID_ ARCHIVE_ HANDLE_ ENTRY	VSID_ ARCHIVE_ HANDLE_ TABLE	VSID_ ARCHIVE_ NAME	VSID_ COMPONENT _HANDLE	VSID_ COMPONENT _HANDLE_ ENTRY
VSCMD_Archive-Query				X	X	X			
VSCMD_Archive-Vary							X		
VSCMD_Audit	X	X	X						
VSCMD_Cancel									
VSCMD_Checkin									
VSCMD_Checkout									
VSCMD_ClearEject									
VSCMD_Connect									
VSCMD_Connect-Query									
VSCMD_Create-Archive MediaClass							X	X	X
VSCMD_Create-MediaClass									
VSCMD_CreatePool									
VSCMD_Delete-Archive MediaClass							X		
VSCMD_Delete-MediaClass									
VSCMD_DeletePool									
VSCMD_Disconnect									
VSCMD_Dismount									
VSCMD_DrivePool Query									
VSCMD_DriveQuery									
VSCMD_DriveVary									
VSCMD_Export									
VSCMD_Import									

	VSID_ ACTION_ CODE	VSID_ ACTION_ CODE_ENTRY	VSID_ ACTION_ CODE_TABLE	VSID_ ARCHIVE_ HANDLE	VSID_ ARCHIVE_ HANDLE_ ENTRY	VSID_ ARCHIVE_ HANDLE_ TABLE	VSID_ ARCHIVE_ NAME	VSID_ COMPONENT _HANDLE	VSID_ COMPONENT _HANDLE_ ENTRY
VSCMD_Intransit-Query									
VSCMD_Lock									
VSCMD_MediaClass Query									
VSCMD_Media-Query									
VSCMD_MediaType Query									
VSCMD_Modify-ArchiveMediaClass							X	X	X
VSCMD_Modify-Media									
VSCMD_Modify-MediaClass									
VSCMD_ModifyPool									
VSCMD_Mount									
VSCMD_Move									
VSCMD_MultiMount									
VSCMD_Ping									
VSCMD_Query-Mount									
VSCMD_Reclassify									
VSCMD_Reprioritize									
VSCMD_Request-Query									
VSCMD_Unlock									

Table A-2

VSID_COMPONENT_HANDLE_TABLE through VSID_DRIVE_HANDLE

	VSID_ COMPONENT _HANDLE_ TABLE	VSID_ COMP_ ID	VSID_COMP _ID_ENTRY	VSID_COM P_ID_ TABLE	VSID_ COMP_STATE	VSID_ CONNECT _HANDLE	VSID_ CONNECT_ HANDLE_ ENTRY	VSID_ CONNECT_ HANDLE_ TABLE	VSID_DRIVE_ HANDLE
VSCMD_Archive-Query									
VSCMD_Archive-Vary					X				
VSCMD_Audit		X	X	X					
VSCMD_Cancel									
VSCMD_Checkin									
VSCMD_Checkout									
VSCMD_ClearEject									
VSCMD_Connect									
VSCMD_Connect-Query						X	X	X	
VSCMD_Create-Archive MediaClass	X								
VSCMD_Create-MediaClass									
VSCMD_CreatePool									
VSCMD_Delete-Archive MediaClass									
VSCMD_Delete-MediaClass									
VSCMD_DeletePool									
VSCMD_Disconnect									
VSCMD_Dismount									
VSCMD_DrivePool Query									
VSCMD_DriveQuery									X
VSCMD_DriveVary									
VSCMD_Export									
VSCMD_Import									

	VSID_ COMPONENT _HANDLE_ TABLE	VSID_ COMP_ ID	VSID_COMP _ID_ENTRY	VSID_COM P_ID_ TABLE	VSID_ COMP_STATE	VSID_ CONNECT _HANDLE	VSID_ CONNECT_ HANDLE_ ENTRY	VSID_ CONNECT_ HANDLE_ TABLE	VSID_DRIVE_ HANDLE
VSCMD_Intransit-Query									
VSCMD_Lock									
VSCMD_MediaClass Query									
VSCMD_Media-Query									
VSCMD_MediaType Query									
VSCMD_Modify-Archive MediaClass	X								
VSCMD_Modify-Media									
VSCMD_Modify-MediaClass									
VSCMD_ModifyPool									
VSCMD_Mount									
VSCMD_Move									
VSCMD_MultiMount									
VSCMD_Ping									
VSCMD_Query-Mount									
VSCMD_Reclassify									
VSCMD_Reprioritize									
VSCMD_Request-Query									
VSCMD_Unlock									

Table A-3

VSID_DRIVE_HANDLE_ENTRY through DSID_DRIVEPOOL_NAME

	VSID_DRIVE_HANDLE_ENTRY	VSID_DRIVE_HANDLE_TABLE	VSID_DRIVE_ID	VSID_DRIVE_ID_ENTRY	VSID_DRIVE_ID_TABLE	VSID_DRIVEPOOL_HANDLE	VSID_DRIVEPOOL_HANDLE_ENTRY	VSID_DRIVEPOOL_HANDLE_TABLE	VSID_DRIVE POOL_NAME
VSCMD_Archive-Query									
VSCMD_Archive-Vary									
VSCMD_Audit									
VSCMD_Cancel									
VSCMD_Checkin									
VSCMD_Checkout									
VSCMD_ClearEject									
VSCMD_Connect									
VSCMD_Connect-Query									
VSCMD_Create-Archive MediaClass									
VSCMD_Create-MediaClass									
VSCMD_CreatePool			X	X	X				X
VSCMD_Delete-Archive MediaClass									
VSCMD_Delete-MediaClass									
VSCMD_DeletePool									X
VSCMD_Disconnect									
VSCMD_Dismount			X	X	X				
VSCMD_DrivePool Query						X	X	X	
VSCMD_DriveQuery	X	X							
VSCMD_DriveVary			X	X	X				
VSCMD_Export									
VSCMD_Import									

	VSID_DRIVE_HANDLE_ENTRY	VSID_DRIVE_HANDLE_TABLE	VSID_DRIVE_ID	VSID_DRIVE_ID_ENTRY	VSID_DRIVE_ID_TABLE	VSID_DRIVEPOOL_HANDLE	VSID_DRIVEPOOL_HANDLE_ENTRY	VSID_DRIVEPOOL_HANDLE_TABLE	VSID_DRIVE POOL_NAME
VSCMD_Intransit-Query									
VSCMD_Lock			X	X	X				
VSCMD_MediaClass Query									
VSCMD_Media-Query									
VSCMD_MediaType Query									
VSCMD_Modify-Archive MediaClass									
VSCMD_Modify-Media									
VSCMD_Modify-MediaClass									
VSCMD_ModifyPool			X	X	X				X
VSCMD_Mount			X	X	X				
VSCMD_Move									
VSCMD_MultiMount			X	X	X				
VSCMD_Ping									
VSCMD_Query-Mount			X	X	X				
VSCMD_Reclassify									
VSCMD_Reprioritize									
VSCMD_Request-Query									
VSCMD_Unlock			X	X	X				

Table A-4

VSID_ERROR_CODE through VSID_MEDIA_HANDLE

	VSID_ ERROR_ CODE	VSID_ ERROR_ CODE_ENTRY	VSID_ ERROR_ CODE_ TABLE	VSID_FIELD	VSID_FIELD_ ENTRY	VSID_FIELD_ TABLE	VSID_ LOCK_ID	VSID_MEDIA_ CLASS_NAME	VSID_MEDIA _HANDLE
VSCMD_Archive-Query									
VSCMD_Archive-Vary									
VSCMD_Audit	X	X	X						
VSCMD_Cancel									
VSCMD_Checkin	X	X	X						
VSCMD_Checkout	X	X	X						
VSCMD_ClearEject	X	X	X						
VSCMD_Connect									
VSCMD_Connect-Query	X	X	X						
VSCMD_Create-Archive MediaClass	X	X	X					X	
VSCMD_Create-MediaClass								X	
VSCMD_CreatePool	X	X	X						
VSCMD_Delete-Archive MediaClass								X	
VSCMD_Delete-MediaClass								X	
VSCMD_DeletePool									
VSCMD_Disconnect									
VSCMD_Dismount							X		
VSCMD_DrivePool Query									
VSCMD_DriveQuery	X	X	X						
VSCMD_DriveVary	X	X	X						
VSCMD_Export	X	X	X						
VSCMD_Import	X	X	X						

	VSID_ ERROR_ CODE	VSID_ ERROR_ CODE_ENTRY	VSID_ ERROR_ CODE_ TABLE	VSID_FIELD	VSID_FIELD_ ENTRY	VSID_FIELD_ TABLE	VSID_ LOCK_ID	VSID_MEDIA_ CLASS_NAME	VSID_MEDIA _HANDLE
VSCMD_Intransit-Query									
VSCMD_Lock							X		
VSCMD_MediaClass Query									
VSCMD_Media-Query	X	X	X						X
VSCMD_MediaType Query	X	X	X						
VSCMD_Modify-Archive MediaClass	X	X	X					X	
VSCMD_Modify-Media	X	X	X	X	X	X			
VSCMD_Modify-MediaClass								X	
VSCMD_ModifyPool	X	X	X						
VSCMD_Mount									
VSCMD_Move	X	X	X						
VSCMD_MultiMount	X	X	X						
VSCMD_Ping									
VSCMD_Query-Mount									
VSCMD_Reclassify	X	X	X						
VSCMD_Reprioritize									
VSCMD_Request-Query	X	X	X						
VSCMD_Unlock	X	X	X				X		

Table A-5

VSID_MEDIA_HANDLE through VSID_MEDIATYPE_HANDLE

	VSID_MEDIA_HANDLE_ENTRY	VSID_MEDIA_HANDLE_TABLE	VSID_MEDIA_ID	VSID_MEDIA_ID_ENTRY	VSID_MEDIA_ID_TABLE	VSID_MEDIACLASS_HANDLE	VSID_MEDIACLASS_HANDLE_ENTRY	VSID_MEDIACLASS_HANDLE_TABLE	VSID_MEDIATYPE_HANDLE
VSCMD_Archive-Query									
VSCMD_Archive-Vary									
VSCMD_Audit			X	X	X				
VSCMD_Cancel									
VSCMD_Checkin			X	X	X				
VSCMD_Checkout			X	X	X				
VSCMD_Clear-Eject			X	X	X				
VSCMD_Connect									
VSCMD_Connect Query									
VSCMD_Create-Archive MediaClass									
VSCMD_Create-MediaClass									
VSCMD_Create-Pool									
VSCMD_Delete-Archive MediaClass									
VSCMD_Delete-MediaClass									
VSCMD_Delete-Pool									
VSCMD_Disconnect									
VSCMD_Dismount			X	X	X				
VSCMD_Drive-Pool Query									
VSCMD_Drive-Query									

	VSID_MEDIA_HANDLE_ENTRY	VSID_MEDIA_HANDLE_TABLE	VSID_MEDIA_ID	VSID_MEDIA_ID_ENTRY	VSID_MEDIA_ID_TABLE	VSID_MEDIACLASS_HANDLE	VSID_MEDIACLASS_HANDLE_ENTRY	VSID_MEDIACLASS_HANDLE_TABLE	VSID_MEDIATYPE_HANDLE
VSCMD_Drive-Vary									
VSCMD_Export			X	X	X				
VSCMD_Import			X	X	X				
VSCMD_Intransit Query			X	X	X				
VSCMD_Lock									
VSCMD_Media-Class Query						X	X	X	
VSCMD_Media-Query	X	X							
VSCMD_Media-Type Query									X
VSCMD_Modify-Archive MediaClass									
VSCMD_ModifyMedia			X	X	X				
VSCMD_Modify-MediaClass									
VSCMD_Modify-Pool									
VSCMD_Mount			X	X	X				
VSCMD_Move			X	X	X				
VSCMD_Multi-Mount			X	X	X				
VSCMD_Ping									
VSCMD_Query-Mount									
VSCMD_Reclassify			X	X	X				
VSCMD_Re-prioritize									
VSCMD_Request Query									
VSCMD_Unlock									

Table A-6

VSID_STATUS_CODE through VSID_WAIT_REASON

	VSID_STATUS_CODE	VSID_STATUS_TYPE	VSID_TARGET_ENTERPRISE_ID	VSID_USER_FIELD	VSID_WAIT_REASON
VSCMD_ArchiveQuery	X	X		X	
VSCMD_ArchiveVary	X	X		X	
VSCMD_Audit	X	X		X	X
VSCMD_Cancel	X	X		X	
VSCMD_Checkin	X	X		X	
VSCMD_Checkout	X	X		X	
VSCMD_ClearEject	X	X		X	
VSCMD_Connect	X	X	X	X	
VSCMD_ConnectQuery	X	X		X	
VSCMD_CreateArchiveMediaClass	X	X		X	
VSCMD_CreateMediaClass	X	X		X	
VSCMD_CreatePool	X	X		X	
VSCMD_DeleteArchiveMediaClass	X	X		X	
VSCMD_DeleteMediaClass	X	X		X	
VSCMD_DeletePool	X	X		X	
VSCMD_Disconnect	X	X	X	X	
VSCMD_Dismount	X	X		X	
VSCMD_DrivePoolQuery	X	X		X	
VSCMD_DriveQuery	X	X		X	
VSCMD_DriveVary	X	X		X	
VSCMD_Export	X	X		X	
VSCMD_Import	X	X		X	
VSCMD_IntransitQuery	X	X		X	
VSCMD_Lock	X	X		X	X
VSCMD_MediaClassQuery	X	X		X	
VSCMD_MediaQuery	X	X		X	
VSCMD_MediaTypeQuery	X	X		X	
VSCMD_ModifyArchiveMediaClass	X	X		X	

	VSID_STATUS_CODE	VSID_STATUS_TYPE	VSID_TARGET_ENTERPRISE_ID	VSID_USER_FIELD	VSID_WAIT_REASON
VSCMD_ModifyMedia	X	X		X	
VSCMD_ModifyMediaClass	X	X		X	
VSCMD_ModifyPool	X	X		X	
VSCMD_Mount	X	X		X	X
VSCMD_Move	X	X		X	
VSCMD_MultiMount	X	X		X	X
VSCMD_Ping	X	X		X	
VSCMD_QueryMount	X	X		X	
VSCMD_Reclassify	X	X		X	
VSCMD_Reprioritize	X	X		X	
VSCMD_RequestQuery	X	X		X	
VSCMD_Unlock	X	X		X	

Table A-7

VSID_MEDIATYPE_HANDLE_ENTRY through VSID_SEQUENCE_TABLE

	VSID_ MEDIATYPE_ HANDLE_ ENTRY	VSID_ MEDIATYPE_ HANDLE_ TABLE	VSID_ PID	VSID_QUERY_ ENTERPRISE_ ID	VSID_ QUERY_ OPTION	VSID_ REQUEST_ HANDLE	VSID_ REQUEST_ ID	VSID_ SEQUENCE _NUM	VSID_ SEQUENCE _TABLE
VSCMD_Archive-Query					X			X	X
VSCMD_Archive-Vary								X	X
VSCMD_Audit								X	X
VSCMD_Cancel							X	X	X
VSCMD_Checkin								X	X
VSCMD_Checkout								X	X
VSCMD_ClearEject								X	X
VSCMD_Connect								X	X
VSCMD_Connect-Query				X				X	X
VSCMD_Create-ArchiveMediaClass								X	X
VSCMD_Create-MediaClass								X	X
VSCMD_CreatePool								X	X
VSCMD_Delete-ArchiveMediaClass								X	X
VSCMD_Delete-MediaClass								X	X
VSCMD_DeletePool								X	X
VSCMD_Disconnect								X	X
VSCMD_Dismount								X	X
VSCMD_DrivePool Query					X			X	X
VSCMD_DriveQuery								X	X
VSCMD_DriveVary								X	X
VSCMD_Export								X	X
VSCMD_Import								X	X

	VSID_ MEDIATYPE_ HANDLE_ ENTRY	VSID_ MEDIATYPE_ HANDLE_ TABLE	VSID_ PID	VSID_QUERY_ ENTERPRISE_ ID	VSID_ QUERY_ OPTION	VSID_ REQUEST_ HANDLE	VSID_ REQUEST_ ID	VSID_ SEQUENCE _NUM	VSID_ SEQUENCE _TABLE
VSCMD_Intransit-Query								X	X
VSCMD_Lock								X	X
VSCMD_MediaClass Query					X			X	X
VSCMD_Media-Query								X	X
VSCMD_MediaType Query	X	X						X	X
VSCMD_Modify-ArchiveMediaClass								X	X
VSCMD_Modify-Media								X	X
VSCMD_Modify-MediaClass								X	X
VSCMD_ModifyPool								X	X
VSCMD_Mount								X	X
VSCMD_Move								X	X
VSCMD_MultiMount								X	X
VSCMD_Ping			X					X	X
VSCMD_Query-Mount								X	X
VSCMD_Reclassify								X	X
VSCMD_Reprioritize							X	X	X
VSCMD_Request-Query						X		X	X
VSCMD_Unlock								X	X

NOTES

NOTES

B

Error Codes

Roadmap

Topic	Refer To Chapter
Naming conventions. Global parameters. Error handling.	1
API functions.	2
Valid staus fields.	A
Error codes.	B
Mount example.	C

Note

Some errors are specific to VSADM functionality, thus they are not marked as being returned for a Client Command.

Error Codes

Code	Definition	Description
0	VSE_VOLERR_NONE	No error.
1	VSE_VOLERR_AUTH	Authorization error. Client is not authorized to send commands.
2	VSE_VOLERR_VERS	Version error. Software versions are incompatible.
3	VSE_VOLERR_PROC	Procedure error.
4	VSE_VOLERR_DB	Database error.
5	VSE_VOLERR_SWCOMM	Software communication error. VolServ cannot communicate with one of its tasks or with an archive.
6	VSE_VOLERR_SYSTEM	System error. Internal system object cannot be found in memory lists. (VolServ may need to be cycled.)
7	VSE_VOLERR_NOTEMPTY	Item not empty.
8	VSE_VOLERR_NOTFOUND	Item not found. Query could not find a match for the given criteria.
9	VSE_VOLERR_EXIST	Item already exists.
10	VSE_VOLERR_INVAL	Invalid item.
11	VSE_VOLERR_BUSY	Device is busy.
12	VSE_VOLERR_OVERFLOW	Overflow.
13	VSE_VOLERR_INVALID_ARCHIVE	Invalid archive. Specified archive does not exist or could not be found. Medium is found, but it is not in the correct archive.

Code	Definition	Description
14	VSE_VOLERR_INVALID_BATCH	Invalid batch. Specified batch could not be found or does not exist.
15	VSE_VOLERR_INVALID_CLASS	Invalid class. Specified MediaClass name could not be found or does not exist.
16	VSE_VOLERR_INVALID_OPT	Invalid option specified. An invalid drive pool option (add/delete) was specified.
17	VSE_VOLERR_EMPTY_LIST	No list specified. No archive/drive/medium list was specified for command requiring list.
18	VSE_VOLERR_BAD_LIST	Error retrieving list items. Query failed to return a valid archive/drive/medium list.
19	VSE_VOLERR_DRIVE_ASSOCIATED	Drive still associated with archive.
20	VSE_VOLERR_IMPORT	Medium failed import.
21	VSE_VOLERR_CHECKIN	Medium failed checkin.
22	VSE_VOLERR_INVALID_COMMAND	Invalid command. Command to cancel/reprioritize was not a valid command type.
23	VSE_VOLERR_DRIVE_IN_POOL	Drive still member of drive pool group.
24	VSE_VOLERR_LIST	Error in list - check individual error codes on item in list. At least one of the items in the list returned an error (i.e., reclassify may be successful for one medium but not another).
25	VSE_VOLERR_EMPTY_POOL	Drive pool empty.
26	VSE_VOLERR_DRIVE_NOT_ASSOCIATED	Drive not associated with an archive.

Code	Definition	Description
27	VSE_VOLERR_INVAL_STATE	Invalid action or location state for operation. The medium is not in the correct state for the specified operation (a medium mount may fail because the medium is marked for export). The archive is not in the correct state to be started/stopped.
28	VSE_VOLERR_INVALID_DRIVE	Invalid drive specified. Specified drive could not be found or is out of valid drive ID range. No drive in the drive list or drive pool is suitable to satisfy the request.
29	VSE_VOLERR_INVALID_MEDIA	Invalid media specified. A medium specified for dismount does not match the medium actually mounted on the specified drive.
30	VSE_VOLERR_INVALID_POOL	Invalid pool specified. Specified drive pool could not be found.
31	VSE_VOLERR_ARCHIVE_MANAGER	Archive manager general error. Components in the archive are unavailable, so requests cannot be processed. Archive or archive components cannot be initialized. Cannot find component to complete request. See logs for more information.
32	VSE_VOLERR_MEM_ALLOC	Memory allocation error. Out of memory.
33	VSE_VOLERR_INVALID_ ATTRIBUTES	Invalid parameters specified.
34	VSE_VOLERR_OPERATOR_FAIL	Operator failed command.

Code	Definition	Description
35	VSE_VOLERR_MEDIUM_IN_ ARCHIVE	Medium already exists in an archive. Medium was found in a different archive, and thus may not be entered into this archive.
36	VSE_VOLERR_UNKNOWN_MEDIUM	Unknown medium. Medium is not known to VolServ software (i.e., it has not been imported).
37	VSE_VOLERR_MUST_CHECKIN_ MEDIUM	Medium must be checked in.
38	VSE_VOLERR_PORT_IN_USE	Specified port is in use.
39	VSE_VOLERR_MEDIUM_TO_BE_ CHECKED_OUT	Medium is to be checked out.
40	VSE_VOLERR_MEDIUM_NOT_TO_ BE_ENTERED	Medium not scheduled to be entered. Some other action is to be taken on the medium; it may be destined for checkout.
41	VSE_VOLERR_ARCHMEDIA_ NOTFOUND	Medium not found in archive. Cannot find the medium in the specified archive.
42	VSE_VOLERR_MEDIUM_DUPLICATE	Duplicate medium. Two media with the same Media ID were found in the archive.
43	VSE_VOLERR_MEDIUM_MOUNTED	Medium mounted.
44	VSE_VOLERR_MEDIUM_NOT_ MOUNTED	Medium not mounted.
45	VSE_VOLERR_MEDIUM_MARKED_ EJECT	Medium already marked for ejection.
46	VSE_VOLERR_MEDIUM_NOT_ MARKED_EJECT	Medium not marked for ejection.

Code	Definition	Description
47	VSE_VOLERR_MEDIA_NOT_ AVAILABLE	Media not available for use. The medium is currently in an unavailable component (i.e., it may be in a CLM that has been taken off-line). The robots that can access the medium are off-line.
48	VSE_VOLERR_MEDIUM_NOT_IN_ SPECIFIED_CLASS	Medium not in specified MediaClass group.
49	VSE_VOLERR_ARCHIVE_CLASS_ NOT_FOUND	Archive not associated with MediaClass group.
50	VSE_VOLERR_ARCHIVE_CLASS_ EXISTS	Archive already associated with MediaClass group.
51	VSE_VOLERR_ARCHIVE_CLASS_IS_ IMPORT_DEFAULT	MediaClass group is the auto-import default for archive.
52	VSE_VOLERR_ARCHIVE_CLASS_ PLACEMENT_NOT_SUPPORTED	MediaClass group placement not supported for this archive type.
53	VSE_VOLERR_INVALID_LOCATION	Invalid archive media class placement location.
54	VSE_VOLERR_ARCHIVE_ MEDIATYPE_NOTFOUND	Archive does not support media type.
55	VSE_VOLERR_ARCHIVE_FULL	Archive has reached capacity for media type. There is no more room in the archive for this media type.
56	VSE_VOLERR_INVALID_LOCKID	Invalid lock ID.
57	VSE_VOLERR_DRIVE_LOCKED	Drive currently locked.
58	VSE_VOLERR_DRIVE_NOT_LOCKED	Drive not locked.
59	VSE_VOLERR_DRIVE_NOT_ AVAILABLE	Drive not available for use.
60	VSE_VOLERR_DRIVE_ALREADY_ ASSOCIATED	Drive already associated.

Code	Definition	Description
61	VSE_VOLERR_DRIVE_ASSOCIATED_WITH_ANOTHER_ARCHIVE	Drive associated with another archive.
62	VSE_VOLERR_ARCHIVE_HW	Archive hardware error.
63	VSE_VOLERR_CMD_NOT_CANCELLABLE	Command not in state to cancel.
64	VSE_VOLERR_CMD_CANCELLED	Command has been cancelled.
65	VSE_VOLERR_CMD_NOTFOUND	Command not found. Unable to cancel or reprioritize.
66	VSE_VOLERR_AUDIT_FAIL	Audit failed.
67	VSE_VOLERR_NEW_MEDIUM	New medium - imported.
68	VSE_VOLERR_UNREADABLE_LABEL	Unreadable medium label.
69	VSE_VOLERR_WRONG_BIN	Medium in wrong bin.
70	VSE_VOLERR_COMP_NOT_SUPPORTED	Component not supported.
71	VSE_VOLERR_COMP_INVALID	Invalid component.
72	VSE_VOLERR_COMPSTATE_INVALID	Invalid component state.
73	VSE_VOLERR_COMP_NOT_AVAILABLE	Component not available.
74	VSE_VOLERR_COMP_NOTFOUND	Component not found.
75	VSE_VOLERR_COMPTYPE_INVALID	Invalid component type.
76	VSE_VOLERR_COMP_DOES_NOT_SUPPORT_MEDIATYPE	Component does not support media type.
77	VSE_VOLERR_RECLASSIFY	Reclassify error.
78	VSE_VOLERR_ROBOT_OFFLINE	Robot off-line.
79	VSE_VOLERR_DRIVE_MOUNTED	Drive is mounted.
80	VSE_VOLERR_PORT_EMPTY	Port is empty.
81	VSE_VOLERR_DRIVE_NOT_MOUNTED	Drive is not mounted.

Code	Definition	Description
82	VSE_VOLERR_ARCHIVE_CLASSES_STILL_EXIST	MediaClass group associated with archives.
83	VSE_VOLERR_MOUNT_RECLASSIFY_NOT_SUPPORTED	Reclassify only supported for mount by MediaClass group.
84	VSE_VOLERR_ARCHIVE_NOT_ONLINE	Archive not on-line.
85	VSE_VOLERR_MEDIA_NOT_IN_LIST	Media is not on list.
86	VSE_VOLERR_MEDIA_ALREADY_IN_LIST	Media is already on list.
87	VSE_VOLERR_MEDIA_NOT_PROCESSED	Media was not processed. Processing, as specified by the request, was not completed for this medium.
88	VSE_VOLERR_MEDIA_ASSIGNED	Media assigned. The specified medium is assigned to be mounted (Query Mount). VolServ tried to assign a medium that was already assigned. If this is the case, there is a VolServ software error (Mount).
89	VSE_VOLERR_MEDIA_NOT_ASSIGNED	Media is not assigned.
90	VSE_VOLERR_INVALID_MANUFACTURER_NAME	Invalid manufacturer name.
91	VSE_VOLERR_INVALID_REQUEST_ID	Invalid request ID. Specified request ID was not a positive non-zero number.
92	VSE_VOLERR_INVALID_PRIORITY	Invalid priority. The range of valid priorities is 1 to 32, inclusive.
93	VSE_VOLERR_INVALID_DRIVE_TYPE	Invalid drive type. The specified drive type is invalid (creation of a drive).

Code	Definition	Description
94	VSE_VOLERR_INVALID_MEDIA_TYPE	Invalid media type. The specified media type was not a valid system or user-defined media type.
95	VSE_VOLERR_INVALID_DRIVE_SELECT	Invalid drive select option. A drive select option other than by drive ID, drive list, or drive pool was specified. Note: Selecting by list is not valid for a mount command.
96	VSE_VOLERR_INVALID_MEDIA_SELECT	Invalid media select option. A media select option other than by media ID, media list, or MediaClass group was specified.
97	VSE_VOLERR_INVALID_CLASS_MOUNT_STATE	Invalid class mount state. The MediaClass group is not defined as mountable by class.
98	VSE_VOLERR_INVALID_ARCHIVE_MODE	Invalid archive mode The specified archive mode is not in the range of allowable modes (attended/unattended).
99	VSE_VOLERR_CLM_ASSOCIATED	Component already associated. The specified component is already associated with a drive.
100	VSE_VOLERR_EJECTION_CLEARED	Medium ejection cleared. A medium marked for ejection (on the Eject list) was unmarked (removed from the Eject list).
101	VSE_VOLERR_MANUAL_EJECT	Medium manually ejected. The specified medium was manually ejected while processing this request.
102	VSE_VOLERR_INVALID_ARCHIVE_ACTION_OPTION	Invalid archive action. The specified archive action option is not within the allowed range (none, migrate, notify).

Code	Definition	Description
103	VSE_VOLERR_INVALID_MIGRATION_PRIORITY	Invalid migration priority.
104	VSE_VOLERR_INVALID_HIGH_MARK	Invalid high mark.
105	VSE_VOLERR_INVALID_LOW_MARK	Invalid low mark.
106	VSE_VOLERR_INVALID_CAPACITY	Invalid capacity.
107	VSE_VOLERR_INVALID_TARGET_ARCHIVE	Invalid target archive. Migration was specified as the archive action option, but an unknown or invalid migration archive was specified.
108	VSE_VOLERR_COMP_OVERLAP	Component overlaps with another in the list. A component specified for preferred placement overlaps with another already in the list (i.e., if row 1,0,0,0 was specified, as was column 1,2,0,0, then the column would return this error as it is redundant).
109	VSE_VOLERR_DRIVE_NOT_ONLINE	Drive not on-line.
110	VSE_VOLERR_MOUNT_CROSSES_ARCHIVES	Mount crosses archives. The mount requires a medium be moved between archives, but the move-wait option specifies no wait.
111	VSE_VOLERR_CLASS_NOT_MOUNTABLE	Class mount state not set to mount. The specified MediaClass group is not defined for mounting by class.
112	VSE_VOLERR_CLASS_CANNOT_STATISFY_MOUNT	No media in class or no drives to satisfy mount.
113	VSE_VOLERR_NO_MEDIA_FOUND_IN_CLASS_FOR_MOUNT	No media available in class to satisfy mount.
114	VSE_VOLERR_NO_DRIVES_FOUND_FOR_MOUNT	No drives available to satisfy Mount.
115	VSE_VOLERR_NO_DRIVES_AVAILABLE_IN_POOL	No drives available in drive pool to satisfy Mount.

Code	Definition	Description
116	VSE_VOLERR_INCONSISTENCY	Software or database inconsistency. Database inconsistency was encountered.
117	VSE_VOLERR_MEDIA_STILL_IN_ CLASS	Media still in class. MediaClass group cannot be deleted, unless there are no media in the class.
118	VSE_VOLERR_DRIVE_ARCHIVE_ INCOMPATIBLE_MEDIA_TYPES	Drive and archive are of incompatible media types. The drive to be associated does not match any media types that the archive supports.
119	VSE_VOLERR_INVALID_MEDIA_ SYNTAX	Invalid media syntax.
120	VSE_VOLERR_INVALID_DRIVE_ SYNTAX	Invalid drive syntax.
121	VSE_VOLERR_CLASS_STILL_ TARGET_CLASS_FOR_MIGRATION	Class still specified as target for migration. The specified archive MediaClass group association cannot be deleted because this archive and MediaClass group are the migration target for another archive media class association.
122	VSE_VOLERR_NOT_ENOUGH_ AVAILABLE_DRIVES_IN_POOL	Not enough available drives in pool. There are not enough available drives in the pool to satisfy the request.
123	VSE_VOLERR_INVALID_RPC_ OPTION	Invalid RPC option.
124	VSE_VOLERR_INVALID_RPC_HOST NAME	Invalid RPC hostname.
125	VSE_VOLERR_INVALID_RPC_ PROGRAM_NUMBER	Invalid RPC program number.
126	VSE_VOLERR_INVALID_RPC_ VERSION	Invalid RPC version.

Code	Definition	Description
127	VSE_VOLERR_INVALID_RPC_PROC_NUMBER	Invalid RPC procedure number.
128	VSE_VOLERR_INVALID_RPC_PROTOCOL	Invalid RPC protocol.
129	VSE_VOLERR_LOCK_CROSSES_ARCHIVES	Lock crosses archives. Only drives in the same archive can be locked with a single lock command.
130	VSE_VOLERR_SOFTWARE_NOT_READY	Software not ready. VolServ is not ready to process requests. It is still initializing.
131	VSE_VOLERR_NOT_ACCEPTING_COMMANDS	VolServ not accepting commands. VolServ is in single-user mode (i.e., it is not accepting commands through the client interface).
132	VSE_VOLERR_COMMAND_NOT_ALLOWED	Command not allowed. The specified command is not allowed over this interface. The system administrator controls which commands are allowed over each interface.
133	VSE_VOLERR_INVALID_LOCK_COUNT	Invalid lock count. Drive lock count is not a positive non-zero number.
134	VSE_VOLERR_NO_AVAILABLE_BINS	No available bins.
135	VSE_VOLERR_INVALID_MODE	Invalid mode. The specified mode is out of range.
136	VSE_VOLERR_COMMAND_IN_UNKNOWN_STATE	Command terminated in unknown state.
137	VSE_VOLERR_MANUFACTURER_MISSING	Manufacturer not specified. A batch identifier was specified without the manufacturer name (none or both are required).

Code	Definition	Description
138	VSE_VOLERR_BATCH_ MISSING	Batch not specified. A manufacturer name was specified with the batch identifier (none or both are required).
139	VSE_VOLERR_CLASS_DOES_NOT_ SUPPORT_MEDIA_TYPE	Class does not support media type.
140	VSE_VOLERR_INVALID_MOVEWAIT_ OPTION	Invalid move-wait option. The move-wait option was not in the range of accepted values (no, yes, attended).
141	VSE_VOLERR_ARCHIVE_ UNATTENDED	Current archive mode is unattended. The request cannot be completed because the source archive was unattended.
142	VSE_VOLERR_TARGET_ARCHIVE_ UNATTENDED	Target archive mode is unattended. The request cannot be completed because the destination archive was unattended.
143	VSE_VOLERR_DELETING_MEDIA_ FROM_CLASS	Error deleting medium from class.
144	VSE_VOLERR_ADDING_MEDIA_TO_ CLASS	Error adding medium to class.
145	VSE_VOLERR_MEDIA_ALREADY_IN _CLASS	Medium already in class. Reclassify failed because medium already exists in MediaClass group.
146	VSE_VOLERR_INVALID_CURRENT_ CLASS	Invalid current class. Reclassify failed because medium does not exist in specified current class.
147	VSE_VOLERR_INVALID_TARGET_ CLASS	Invalid target class. Reclassify failed because target class is not a valid class for this medium.

Code	Definition	Description
148	VSE_VOLERR_MEDIA_STILL_IN_ ARCHIVE	Media still in archive. If environment variables are set appropriately, then archive that still has media in it cannot be deleted.
149	VSE_VOLERR_MIXED_MEDIA_ TYPES_IN_LIST	Media list with mixed media types. Mount failed because the specified media contained media of differing media types.
150	VSE_VOLERR_FILLLEVEL_ EXCEEDS_NEW_CAPACITY	Current fill level exceeds desired capacity. The modify class failed because the current fill level is above the requested capacity.
151	VSE_VOLERR_MAX_MEDIATYPES_ DEFINED	Maximum number of media types defined.
152	VSE_VOLERR_ARCHMEDIA_MOVE_ FAILED	Archive media move failed (not currently returned).
153	VSE_VOLERR_CLASS_CAPACITY_ INCONSISTENCY	Archive media class inconsistency (not currently returned).
154	VSE_VOLERR_MEDIA_CLASS_FULL	MediaClass group is full.
155	VSE_VOLERR_BIN_EMPTY	Bin empty. Medium movement failed because the medium was not in the bin.
156	VSE_VOLERR_BIN_NOT_EMPTY	Bin not empty. Medium movement failed because a medium already exists in the destination bin (several retries with a new bin will be tried before failing).
157	VSE_VOLERR_DRIVE_MEDIA_ ASSIGNED	Drive has media assigned. Mount failed because the drive already had a medium assigned to it.
158	VSE_VOLERR_CLIENT_ CONNECTED	Client connected.

Code	Definition	Description
159	VSE_VOLERR_UNABLE_TO_ CONNECT_CLIENT	Unable to connect to client.
160	VSE_VOLERR_INVALID_ ENTERPRISE_ID	Invalid enterprise ID.
161	VSE_VOLERR_UNKNOWN_ ENTERPRISE_ID	Unknown enterprise ID.
162	VSE_VOLERR_UNKNOWN_CLIENT	Unknown client.
163	VSE_VOLERR_UNABLE_TO_ DISCONNECT_CLIENT	Unable to disconnect from client.
164	VSE_VOLERR_ENTER_WRONG_ ARCHIVE	Medium to be entered in another archive.
165	VSE_VOLERR_CHECKEDIN_ MEDIUM	Medium checked in.
166	VSE_VOLERR_MEDIUM_ID_ MISMATCH	Medium ID specified does not match bar code. Medium movement failed because the wrong medium was in the bin.
167	VSE_VOLERR_COMP_ INCONSISTENT_MAPPINGS	Inconsistency in column mappings. During configuration or reconfiguration, there was an inconsistency in the column mappings.
168	VSE_VOLERR_EU_HAS_ ASSOCIATED_MEDIA	CLM or CLU with associated media - manual eject. This is the result of a Mount failing because the CLM could not move the medium to the recorder, and error recovery processing could not correct the problem.
169	VSE_VOLERR_ARCHIVE_MANAGER _TERMINATING	Archive manager terminating - request cancelled. The archive is in the process of shutting down; therefore, the request has been cancelled.

Code	Definition	Description
170	VSE_VOLERR_MEDIA_NOT_ EJECTED_FROM_DRIVE	Medium not ejected from drive. Dismount failed because the medium was not ejected from the drive.
171	VSE_VOLERR_MEDIA_ALREADY_IN _TARGET_ARCHIVE	Medium already exists in target archive. Move failed because medium already resides in the destination archive.
172	VSE_VOLERR_MEDIUM_NOT_ MOUNTED_ON_DRIVE	Medium is not mounted on specified drive.
173	VSE_VOLERR_PORT_NO_ AVAILABLE_BINS	No available port bins.
174	VSE_VOLERR_PORT_COUNT_TOO_ LARGE	Port count too large.
175	VSE_VOLERR_INVALID_MEDIA_ STAT_VALUE	Invalid media statistic value for user-defined media statistics.
176	VSE_VOLERR_INVALID_MOUNT_ CRITERIA	Invalid mount criteria.
177	VSE_VOLERR_NO_MEDIA_ SELECTED_BY_CRITERIA	No media found that meet user criteria.
178	VSE_VOLERR_INCONSISTENT_ MEDIA_ON_LIST	Media on console list.
179	VSE_VOLERR_CANNOT_ REPRIORITIZE_CMD	Command not in state to reprioritize.
180	VSE_VOLERR_INVALID_DRIVE_LIST	Invalid Drive list.
181	VSE_VOLERR_MEDIUM_IN_PORT	Medium in port.
182	VSE_VOLERR_PROTOCOL_NOT_ SUPPORTED	Protocol not supported at this time.
183	VSE_VOLERR_MEDIUM_NOT_ REMOVED_FROM_ARCHIVE	Medium not physically removed from archive. Manual eject was requested, but medium was not physically removed (only the STK ACS product family archives verify that medium was physically removed).

Code	Definition	Description
184	VSE_VOLERR_ARCHIVE_ PHYSICALLY_FULL	Archive is physically full.
185	VSE_VOLERR_INVALID_SOCKET_ INFORMATION	Invalid socket information.
186	VSE_VOLERR_ROBOT_ACCESSOR_ FULL	Accessor contains a medium.
187	VSE_VOLERR_AUDIT_IN_ PROGRESS	Shelf cannot be varied during an audit.
188	VSE_VOLERR_PORT_NOT_ AVAILABLE	Port is not available.
189	ERR_HARDWARE_NOT_READY	Hardware not ready.
190	ERR_MAC_OFFLINE	Manipulator Accessor Component (MAC) is offline or unavailable.
191	ERR_MAC_NOTFOUND	MAC not found.
192	ERR_PTBIN_OFFLINE	Pass-through bin offline or unavailable.
193	ERR_PTBIN_NOTFOUND	Pass-through bin not found.
194	ERR_ROW_NOTFOUND	Row not found.
195	ERR_CLU_INVALID	Port not accessible or offline.
196	ERR_HARDWARE_COMPONENT_ OFFLINE	Physical hardware component offline.
197	ERR_MEDIUM_CHECKED_OUT	Medium checked out.
198	ERR_MEDIUM_INTRANSIT	Medium intransit.
199	ERR_ARCHIVE_MOVEMENT_ PENDING_FOR_MEDIUM	Archive movement pending for medium.
200	ERR_LICENSE_STRING_MISSING	License string missing.
201	ERR_LICENSE_LIMIT_REACHED	License limit reached.
202	ERR_INVALID_LICENSE_STRING	Invalid license string.
203	ERR_EJECT_IN_PROGRESS	Eject currently in progress for medium.

Code	Definition	Description
204	ERR_TIMEOUT_WAITING_ON_ HARDWARE_STATUS	Timeout waiting on hardware status.
205	ERR_HARDWARE_COMMUNICATION	Hardware communication error.
206	ERR_ARCHIVE_TYPE_NOT_ SUPPORTED	Archive type is not supported.
207	ERR_DISMOUNT_ALREADY_ QUEUED	Medium dismount is already queued.

NOTES

C

Mount Example

Roadmap

Topic	Refer To Chapter
Naming conventions. Global parameters. Error handling.	1
API functions.	2
Valid staus fields.	A
Error codes.	B
Mount example.	C

Example

Mount by MediaClass Group & Drive Pool:

```
1  #include <stdio.h>
2  #include <vs_client.h>
3
4  /* include the VolServ API header file*/
5
6  /* define the volserv host computer */
7  #define VOLSERV_HOST    "vshost"
8
9  /* forward declare the dispatch routine
       */
10 void dispatch ( VST_COMMAND_HANDLE );
11
12 void print_error ( VST_ERROR_HANDLE );
13
14 main ( int argc, char *argv[] )
15 {
16    int      exitcode = 0;
17
18    /* the name of the MediaClass group
       from */
19    /* which to select the medium to
       mount   */
20    VST_MEDIA_CLASS_NAME mediaclass;
21
22    /* the name of the DrivePool group
       from  */
23    /* which to select the drive to mount
       */
24    VST_DRIVE_POOL_NAME  drivepool;
25
26    /* the identifier of the medium that
       was mounted */
27    VST_MEDIA_ID         mediaid;
28
29    /* the identifier of the drive that
       was mounted */
30    VST_DRIVE_ID         driveid;
31
```

```
32    /* the command handle for the Mount
         request */
33    VST_COMMAND_HANDLE   cmdh;
34
35    /* the status handle for the Mount
         request */
36    VST_STATUS_HANDLE    stath;
37
38    /* the error handle for the Mount
         request */
39    VST_ERROR_HANDLE     errh;
40
41    /* the error object code for the
         Mount request */
42    VST_ERROR_OBJCODE errobj;
43
44    /* the numeric error code for the
         Mount request */
45    VST_ERROR_NUMCODE errnum;
46
47    /* check for the proper number of
         arguments */
48    if ( argc != 3 )
49    {
50       printf ("usage: mount
         <mediaclass> <drivepool>\n");
51       exit ( 1 );
52    }
53
54    /* copy the MediaClass name and the
         drive pool name */
55    strcpy ( mediaclass, argv[1] );
56    strcpy ( drivepool, argv[2] );
57
58    /* initilize the VolServ API */
59    if ( VS_Initialize ( VOLSERV_HOST, 0,
         10 ) )
60    {
61
62       /* create a command handle to hold
         the ping */
63       /* and mount request */
```

```
64        if ( (cmdh = VS_Command_Create ())
          !=

          (VST_COMMAND_HANDLE) NULL )
65        {
66          /* check to see if VolServ s/w is
          available */
67             if ( ! VSCMD_Ping ( cmdh ) )
68             {
69                printf ( "VolServ system is
          not available\n"
                         );
70                exit ( 1 );
71             }
72
73
          /*****************************
          ********/
74          /* Send the Mount request.
          */
75          /* The default API mode is
          synchronous.     */
76          /* Therefore, wait for status
          until the     */
77          /* command is complete or until
          it        */
78          /* times-out waiting for status.
          */
79          /* Set the timeout for three
          minutes and    */
80          /* the retry to two so that the
          total time  */
81          /* to wait is nine minutes.
          */
82          /* Also,register a dispatch
          routine to */
83          /* catch mount intermediate
          status.    */
84
          /*****************************
          ********/
85             if ( VSCMD_Mount ( cmdh,
```

```
86              VSID_MEDIA_CLASS_NAME,
        mediaclass,
87              VSID_DRIVEPOOL_NAME,
        drivepool,
88              VSID_TIMEOUT_VALUE,
        180,
89              VSID_RETRY_LIMIT,
        2,
90              VSID_CLIENT_DISPATCH,
        dispatch,
91                 VSID_ENDFIELD ) )
92         {
93            /* Mount was successful.
        */
94            /* Get the status handle
        */
95            VS_Command_GetFields (
        cmdh,
96
        VSID_STATUS_HANDLE,&stath,
97                         VSID_ENDFIELD
        );
98
99         /* Get identifiers of medium
        and drive   */
100        /* that were mounted from
        status handle */
101        VS_Status_GetFields ( stath,
102
        VSID_MEDIA_ID,mediaid,
103
        VSID_DRIVE_ID,&driveid,
104
        VSID_ENDFIELD );
105
106           /* Output user message
        showing */
107           /* identifiers of mounted
        medium and */
108           /* drive. */
```

```
109         printf ( "Media [%s] mounted
        on drive
                        [%d]\n",
        mediaid, driveid );
110      }
111      else
112      {
113
        /*****************************
        *****/
114         /* Mount was not successful.
        */
115         /* Get error handle from
        command to */
116         /* determine where error
        occurred   */
117
        /*****************************
        *****/
118         VS_Command_GetFields (
        cmdh,
119
        VSID_ERROR_HANDLE,&errh,
120
        VSID_ENDFIELD );
121
122
        /*****************************
        *****/
123         /* Get error object and
        numeric code                */
124         /* from error handle.
        */
125         /* The error object code
        tells where                 */
126         /* the error occurred.
        */
127         /* The error numeric code
        tells  what                 */
128         /* error occurred.
        */
```

```
129
        /*****************************
        ******/
130           VS_Error_GetFields ( errh,
131                   VSID_ERROR_NUMBER,
        &errnum,
132                   VSID_ERROR_OBJECT,
        &errobj,
133                   VSID_ENDFIELD );
134
135
        /*****************************
        *****/
136           /* if  command's error
        number states  */
137           /* no error, error is stored
        in the  */
138           /* global error code.
        */
139
        /*****************************
        *****/
140           if ( errnum != VSE_ERR_NONE
        )
141           {
142
143
        /*****************************
        **/
144              /* check to see if error
        was a                    */
145              /* VolServ error. if yes,
        print a                  */
146              /* message to the user
        stating the              */
147              /* error condition.
        */
148
        /*****************************
        **/
149              if ( errobj ==
        VSE_VOLSERV )
```

```
150             {
151
152                 switch ( errnum )
153                 {
154                     case
        VSE_VOLERR_INVALID_CLASS:
155                         printf (
                            "Invalid media
                            class[%s]\n",
                            mediaclass);
156                         break;
157                     case
        VSE_VOLERR_INVALID_POOL:
158                         printf (
        "Invalid drive
                            pool[%s]\n"
        drivepool );
159                         break;
160                     case

        VSE_VOLERR_NO_DRIVES_AVAILABLE_I
        N_POOL:
161                         printf ( "No
        drives
                            available in
        drive pool
                            [%s]\n",
        drivepool );
162                         break;
163                     case
        VSE_VOLERR_CLASS_NOT_MOUNTABLE:
164                         printf ( "Media
        class [%s]
                            does not allow
        mounts\n",
                            mediaclass );
165                         break;
166                     case

        VSE_VOLERR_NO_MEDIA_FOUND_IN_CLA
        SS_FOR_MOUNT:
```

```
167                     printf ( "Media
        class [%s]
                        has no media
        available for

        mounting\n",mediaclass );
168                     break;
169                  default:
170                     printf (
        "VolServ error
                        encountered \n"
        );
171                     break;
172                }
173             }
174
175
        /*****************************
        **/
176            /* print the error code
        for the user */
177
        /*****************************
        **/
178           print_error ( errh );
179           }
180           else
181           {
182
        /*****************************
        **/
183             /* print the error code
        from the                      */
184             /* VolServ global error
        handle                        */
185
        /*****************************
        **/
186             print_error ( VSG_Error
        );
187           }
188
```

```
189            exitcode = 1;
190         }
191      }
192      else
193      {
194
         /*******************************
         *********/
195         /* print the error code from
         the                             */
196         /* VolServ global error handle
         */
197
         /*******************************
         *********/
198         print_error ( VSG_Error );
199         exitcode = 1;
200      }
201   }
202   else
203   {
204
         /*******************************
         *********/
205      /* print the error code from the
         */
206      /* VolServ global error handle
         */
207
         /*******************************
         *********/
208      print_error ( VSG_Error );
209      exitcode = 1;
210   }
211
212   exit ( exitcode );
213}
214
215/*********************************
         ***********/
216/* The dispatch routine registered with
         the mount*/
```

```
/* command. This routine is called by
   the API
   */
/* whenever status (final or
   intermediate) is
   */
/* received for the Mount request. For
   this      */
/* example, we are interested in
   intermediate
   */
/* status. if we receive final status,
   we ignore */
/* it and let the main routine print out
   the    */
/* media/drive used.
   */
/************************************
   ***********/
void
dispatch ( VST_COMMAND_HANDLE cmdh )
{
   VST_STATUS_HANDLE stath;
   VST_STATUS_TYPE   stattype;
   VST_WAIT_REASON   wait;


   /*******************************
   *************/
   /* Get status handle from Mount
   request
   */

   /*******************************
   *************/
   VS_Command_GetFields ( cmdh,

   VSID_STATUS_HANDLE,&stath,
                         VSID_ENDFIELD
   );

```

```
239
        /*****************************
        *************/
240     /* Get status type (intermediate or
        final)   */
241     /* and wait reason (if any exists).
        */
242
        /*****************************
        *************/
243     VS_Status_GetFields ( stath,
244                           VSID_STATUS_TYPE,
        &stattype,
245                           VSID_WAIT_REASON,
        &wait,
246                           VSID_ENDFIELD );
247
248
        /*****************************
        *************/
249     /* If intermediate status, print
        message to      */
250     /*user. If final status, ignore it.
        */
251
        /*****************************
        *************/
252     if ( stattype ==
        VSE_STATUSTYPE_INTERMEDIATE )
253     {
254        /* print message depending upon
        wait reason */
255        switch ( wait )
256        {
257           case VSE_WAIT_LOCKED_DRIVE:
258              printf ( "Mount waiting on
                 locked drive\n" );
259              break;
260           case VSE_WAIT_BUSY_DRIVE:
261              printf ( "Mount waiting on
                 busy drive\n");
262              break;
```

```
263         case VSE_WAIT_BUSY_MEDIUM:
264            printf ( "Mount waiting on
               busy medium\n" );
265            break;
266         default:
267            printf ( "Unknown wait
               reason [%d]
               received\n",wait );
268            break;
269      }
270   }
271}
272
273/* print error code from given error
      handle */
274void
275print_error ( VST_ERROR_HANDLE errh )
276{
277   VST_ERROR_CODE errcode;
278
279   VS_Error_GetFields ( errh,
280                        VSID_ERROR_CODE,
      errcode,
281                        VSID_ENDFIELD );
282
283   printf ( "Error code [%s] returned
      from API\n", errcode );
284}
```

NOTES

NOTES